Lynn Conway

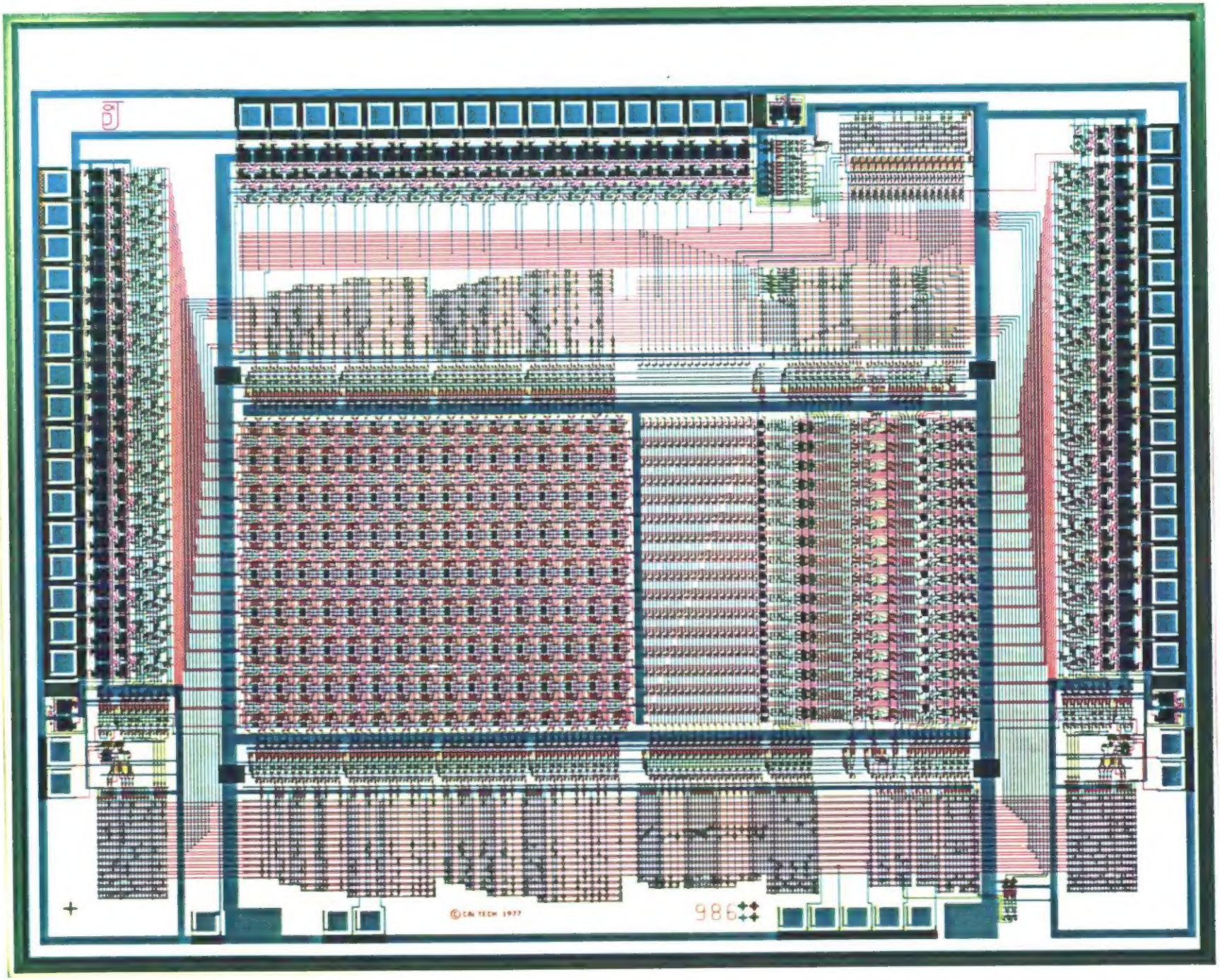
INTRODUCTION TO VLSI SYSTEMS

CARVER A. MEAD

Professor of Computer Science,
Electrical Engineering,
and Applied Physics,
California Institute of Technology

LYNN A. CONWAY

Manager, LSI Systems Area,
Palo Alto Research Center,
Xerox Corporation


©CAL TECH 1977
986

PREFACE:

As a result of improvements in fabrication technology, Large Scale Integrated (LSI) electronic circuitry has become so dense that a single silicon LSI chip may contain tens of thousands of transistors. Many LSI chips, such as microprocessors, now consist of multiple, complex subsystems, and thus are really *integrated systems* rather than integrated circuits.

What we have seen so far is only the beginning. Achievable circuit density approximately doubles with each passing year. How long can this continue, and how small can the transistor be made? From the physics we find that the linear dimensions of transistors can be reduced to less than 1/10 of those in current integrated systems before they cease to function as the sort of switching elements from which we can easily build digital systems. It will eventually be possible to fabricate chips with hundreds of times as many components as today's. The transistors in such very large scale integrated (VLSI) systems will ultimately have linear dimensions smaller than the wave-length of visible light. New, non-optical, high-resolution lithographic techniques are now being developed by many firms to enable fabrication of such circuitry.

The emerging VLSI presents a challenge not only to those involved in development of appropriate fabrication technology, but also to computer scientists and computer architects. The ways in which digital systems are structured, the procedures used to design them, the tradeoffs between hardware and software, and the methodology and metrics of analysis of algorithms will all be greatly affected by the maturation of electronic technology towards its maximum density. We believe this will be an important area of activity for computer science on through the 1980's.

Until recently, the design of integrated electronic circuitry has been largely the province of circuit and logic designers working within semiconductor firms. Computer system architects have traditionally built systems from standard integrated circuits designed and manufactured by these firms, but haven't often participated in the specification and design of these integrated circuits. Electrical Engineering and Computer Science (EE/CS) curricula have reflected this tradition, with courses in device physics and integrated circuit design aimed at and generally taken by different students than those interested in digital system architecture and computer science.

This text is written to fill the current gap by providing students of computer science and electrical engineering with an introduction to integrated system architecture and design. Combined with individual study in related research areas and participation in actual system design projects, the text could serve as a basis for a graduate course sequence in integrated systems. Portions could be used for an undergraduate text on the subject, or to augment a graduate course on computer architecture. It could also be used to extend, in the system direction, a classical electrical engineering course in integrated circuits. We assume the reader's background contains the equivalent of an introductory course sequence in computer science, and introductory courses in electronic circuits and digital design.

Up till now there have been major obstacles in the path of those attempting to gain an overall understanding of integrated systems. Integrated electronics, developing in a heatedly competitive business environment, has proliferated into a large array of different device technologies, circuit design families, logic design techniques, maskmaking and wafer fabrication techniques, etc. The technologies have sprung up from the grass roots of "Silicon Valley" in California. Most participants in the industry have of necessity concentrated on rather narrow specialties. Texts on the subject have tended to give detailed accounts of some very narrow horizontal segment of the overall subject, such as device physics or circuit design.

We have chosen instead to provide the minimum of basic information about devices, circuits, fabrication technology, logic design techniques, and system architecture, which is sufficient to enable the reader to fully span the entire range of abstractions, from the underlying physics to complete VLSI digital computer systems. A rather small set of carefully selected key concepts is all that is necessary for this purpose. We believe that only by carrying along the least amount of unnecessary mental baggage at each step in such a study, will the student emerge with a good overall understanding of the subject. Once this range of abstractions is spanned, the sequence of concepts can then be mapped into the reader's own space of application and technology.

Another major obstacle has been the high rate of change of integrated electronics. The uninitiated could easily get the feeling that much energy could be invested in learning material which becomes obsolete as rapidly as it is assimilated. The major driving mechanism in all this change is the continual improvement in fabrication technology. This evolutionary process results in the feasibility of manufacturing smaller and smaller devices as time passes. By including the effects of scaling down device dimensions as an essential ingredient of all topics in this text, many of the important changes of the architectural parameters of the technology are predicted, expected, and indeed hoped for.

The key concepts of this text are illustrated by way of specific examples. In any given technology, form follows function in a particular way. The skill of mapping function into form, once acquired, can be readily applied to any technology. Because of its density, speed, topological properties, and general availability of wafer fabrication, nMOS has been chosen as the technology in which examples are implemented.

An atmosphere of excitement and anticipation pervades this field. Workers from many backgrounds, computer scientists, electrical engineers, and physicists, are collaborating on a common problem area which has not yet become classical. The territory is vast, and largely unexplored. The rewards are great for those who simply press forward.

Carver A. Mead
Pasadena, California

Lynn A. Conway
Belmont, California

BACKGROUND

Prior to the commercial publication of this textbook, this limited printing of the first five chapters is being distributed to a selected group of universities for use as course notes in graduate courses on integrated systems. Copies are also being distributed to the industrial participants in the Caltech Silicon Structures Project, and to selected individuals in universities and in industry for their review.

The authors welcome any and all comments and suggestions from readers. We are especially interested in hearing of the experiences of those teaching from the text. Notifications and corrections of errors, ideas for improvements in the tutorial techniques used, and suggestions of instructive problems for each chapter would be appreciated.

While the material in this text is presented in a particular order, it need not be read in that order. Each chapter presents material from a distinct level in the hierarchy of disciplines involved in integrated systems. We recommend that the reader start in the chapter where he or she is most knowledgeable, and read until information is required from an adjacent area described in some other chapter. By using this algorithm and consulting the reading references where necessary, the reader can gradually work through the primary material of all chapters. Although much of the material in this text is previously unpublished, it contains only fundamental concepts. However, these concepts cover a wide range of disciplines, and are easily conceptualized only after the overall context becomes clear.

This text has its origins in a series of courses in integrated circuit design given by Carver Mead at Caltech, beginning in 1970. Starting in 1971, students in these courses designed and debugged their own integrated circuits. The students undertook increasingly complex system designs, using only rather simple implementation aids. The structured design methodology presented in this text evolved from this milieu.

A separate Computer Science department was formed at Caltech in 1976, with integrated systems as a focus. An early association was formed with systems architects in industry. Interaction of Caltech students and faculty with industrial researchers resulted in the idea of a joint Caltech-industry cooperative program. The initial industrial participants in this program, known as the Caltech Silicon Structures Project, are Intel, DEC, IBM, Hewlett-Packard, and Xerox.

Work on this text began in August 1977. The first three chapters were used as course notes during the fall of 1977, in courses given by Carver Mead at Caltech and by Carlo Sequin at U. C. Berkeley. This present printing is being used during the spring of 1978 in courses given by Ivan Sutherland at Caltech, by Robert Sproull at Carnegie-Mellon University, and by Fred Rosenberger at Washington University, St. Louis.

ACKNOWLEDGEMENTS

We wish to express our gratitude to the many individuals who are contributing ideas, time, and energy towards the creation of this textbook. In particular we wish to thank the following:

For contributions to the text:

David Johannsen, Caltech, for the OM2 Specification section; Robert Sproull, Carnegie-Mellon, and Richard Lyon, Xerox PARC, for the CIF section; Carlo Sequin, U. C. Berkeley, for his detailed review of the text and his many suggestions for improvements; Hank Smith, M.I.T. Lincoln Laboratory, for his contributions of ideas for the section on x-ray lithography; Douglas Fairbairn, Xerox PARC, and James Rowson, Caltech, for the ICARUS section; Dale Green and his group at Xerox EOS for their technical assitance in preparation of the chapter 5 color plates.

For reviewing drafts:

Robert Sproull; Ivan Sutherland, Caltech; Charles Molnar, Washington University; Chuck Thacker, Wayne Wilner, and Alan Kay, Xerox.

For participation in the OM projects:

Dave Johannsen, Mike Tolle, Chris Carrol, Rod Masumoto, Ivan Sutherland, Chuck Seitz.

For helping to define and establish the multi-project chip capability:

Carl Pompeii, NBK; Joel Sorem, Micro Mask; Dan Izumi and George Marr, Maruman Integrated Circuits; Vir Dhaka, Doug Fairbairn, and Bill Winfield, Xerox; Jim Rowson, Ron Ayres, and Steve Trimberger, Caltech; Bill Lattin and Ted Jenkins, Intel.

For their support since the early days:

The Office of Naval Research;
Robert Noyce and Gordon Moore, Intel Corporation.

We are especially grateful to W. R. Sutherland, Manager, Systems Science Laboratory of Xerox PARC, for his encouragement and support.

Printed by C. Mead and L. Conway,
Palo Alto, California, February, 1978.

TABLE OF CONTENTS

Chapter 1: MOS Devices and Circuits

Sections:

In this chapter we begin with a discussion of the basic properties of the n-channel, metal-oxide-semiconductor (MOS), field effect transistor (FET). We then describe and analyze a number of circuits composed of interconnected MOS field effect transistors. The circuits described are typical of those we will commonly use in the design of integrated systems. The analysis, though highly condensed, is conceptually correct and provides a basis for the solution of most system problems typically encountered.

Integrated systems in MOS technology contain three levels of conducting material separated by intervening layers of insulating material. Proceeding from top to bottom, these levels are termed the *metal*, the *polysilicon*, and the *diffusion* levels respectively. Patterns for paths on these three levels, and the locations of contact cuts through the insulating material to connect certain points between levels, are transferred into the levels during the fabrication process from *masks* similar to photographic negatives. The details of the fabrication process will be discussed in chapter 2.

In the absence of contact cuts through the insulating material, paths on the metal level may cross over paths on the polysilicon or diffusion levels with no significant functional effect. However, wherever a path on the polysilicon level crosses a path on the diffusion level, a transistor is created. Such a transistor has the characteristics of a simple switch, with a voltage on the polysilicon level path controlling the flow of current in the diffusion level path. Circuits composed of such transistors, interconnected by patterned paths on the three levels, form our basic building blocks. With these basic circuits, we will architect integrated systems, to be fabricated on the surface of monolithic crystalline chips of silicon.

The MOS Transistor

An MOS transistor will be produced on the integrated system chip wherever a polysilicon path crosses a diffusion path, as shown in figure 1a. The electrical symbol used to represent the MOS transistor in our circuit diagrams is shown in figure 1b, along with symbols and polarities of certain voltages of interest. Note that the source and drain terminals of the device are physically symmetrical. For the n-channel MOSFETs, these terminal labels are assigned such that V_{ds} is normally positive. A more detailed view of the rectangular region called the gate, where the polysilicon (poly) crosses the diffusion, is given in figure 1c. During fabrication the diffusion paths are formed after the poly paths are formed, as explained more fully in chapter 2. The poly gate and thin layer of oxide under masks the region under the gate during diffusion, thus interrupting the diffusion path under the gate. Therefore, there is no direct diffusion connection between the source and drain terminals of the transistor. Notice in this discussion that metal, poly, and diffusion paths all conduct electricity well enough to be considered "wires" until further notice.

In the absence of any charge on the gate, the drain to source path through the transistor is like an open switch. The gate, separated from the substrate by the layer of thin oxide, forms a capacitor. If sufficient positive charge is placed on the gate so that V_{gs} exceeds a *threshold voltage* V_{th}, electrons will be attracted to the region under the gate to form a conducting path between drain and source. Most of the transistors we will use in our systems have threshold voltages greater than zero. These are called *enhancement mode* MOSFETs, and their threshold voltage typically equals ~ 0.2(VDD), where VDD is the positive supply voltage for the particular technology.

The basic operation performed by the MOS transistor is to use charge on its gate to control the movement of negative charge between source and drain through the channel under the gate. The current from source to drain equals the charge induced in the channel divided by the transit time or average time required for an electron to move from source to drain. The transit time itself is the distance the electron has to move divided by its average velocity. In semiconductors under normal conditions, the velocity is proportional to the electric field driving the electrons. The relationship between drain to source current I_{ds}, drain to source voltage V_{ds}, and gate to source voltage V_{gs} is sketched in figure 1d. For small V_{ds}, the transit time τ is given by equation 1.

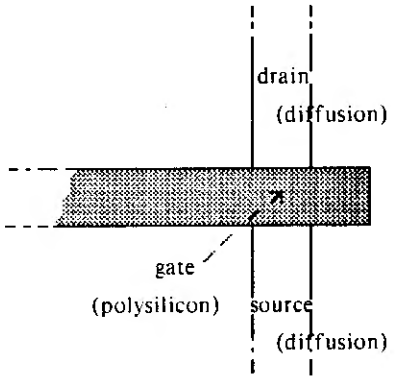


Fig. 1a. MOS Transistor
(top view)

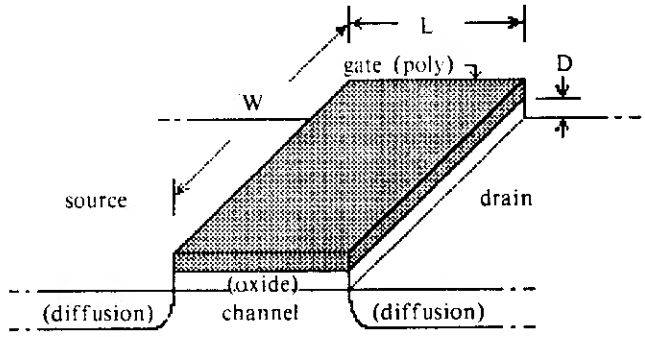


Fig. 1c. MOSFET Gate Dimensions

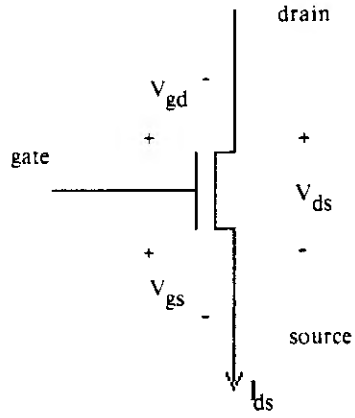


Fig. 1b. MOS Transistor Symbol
(subscripts in + to - direction sequence)

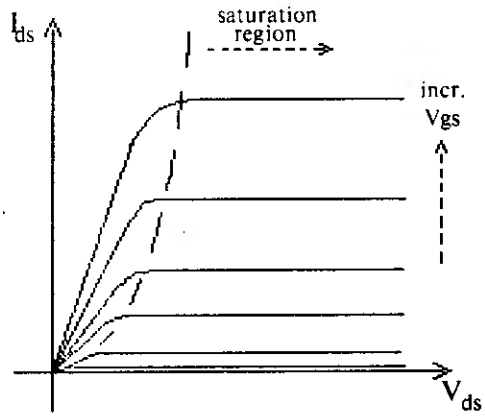


Fig. 1d. Current vs Voltage

Transit time:

$$\tau = L/\text{velocity} = L/\mu E = L^2/[\mu V_{ds}] \qquad \text{[eq.1]}$$

The proportionality constant μ is called the *mobility* of the charge carriers, in this case electrons, under the influence of an electric field in the conducting material of the channel region. It is a velocity per unit electric field (cm^2/volt-sec). We shall see that the transit time is the fundamental time unit of the entire integrated system.

The amount of negative charge in transit is just the gate capacitance times the voltage on the gate in excess of the threshold voltage. The capacitance of two parallel conductors of area A, separated by insulating material of thickness D, equals $\varepsilon A/D$. The proportionality constant ε is called the permittivity of the insulating material, and has a simple interpretation. It is the capacitance of parallel conductors of area A = 1 cm^2, separated by a thickness D = 1 cm of the insulator material, and is in the units farad/cm. Therefore, the gate capacitance equals $\varepsilon WL/D$. Thus the charge in transit is given by eq. 2, and the current is given by eq. 3.

Charge in transit:

$$Q = -C_g(V_{gs} - V_{th}) = -\frac{\varepsilon WL}{D}(V_{gs} - V_{th}) \qquad \text{[eq.2]}$$

Current:

$$I_{ds} = -I_{sd} = -\frac{\text{charge in transit}}{\text{transit time}} = \frac{\mu\varepsilon W}{LD}(V_{gs} - V_{th})(V_{ds}) \qquad \text{[eq.3]}$$

Note that for small V_{ds}, the drain current is proportional to the source-drain voltage and also to the gate voltage above threshold. Any device with a current through it proportional to the voltage across it, may be viewed as a resistor, and in the case of an MOS device with *low* drain to source voltage, the resistance is controlled by the gate voltage as given in eq. 3a.

$$V_{ds}/I_{ds} = R = L^2/[\mu C_g(V_{gs} - V_{th})] \qquad \text{[eq.3a]}$$

In both equations 2 and 3a, C_g is the gate to channel capacitance of the turned on transistor. In the simple case where this transistor is driving the gate of another one identical to it, the time response of the system will be an exponential with a time constant RC_g, given in equation 4. This time constant is identical to the transit time τ given in equation 1.

$$RC_g = L^2/[\mu(V_{gs} - V_{th})] = \tau \qquad \text{[eq.4]}$$

Although the above equations are greatly simplified, they provide sufficient information to make many design decisions which we will face, and also give us insight into the scaling of devices to smaller sizes. In particular, the transit time τ can be viewed as the basic time unit of any system we shall build in the integrated technology. In almost all situations, the fastest operation which we can perform is to transfer a signal from the gate of one MOS transistor onto the gate of another. The transit time is the minimum time in which a charge placed on the gate of one transistor results in the transfer of a similar charge through that transistor's channel onto the gate of a subsequent transistor. For example, to transfer a charge from one transistor onto two transistors identical to it requires a minimum of two transit times. Thus, the transit time of the basic transistor in an integrated system can be viewed as the unit of time in which all other times in the system are scaled. Although it is a somewhat optimistic approximation, we will use τ as the primary time metric in calculating the delay through elementary inverting logic stages. More accurate predictions of circuit behavior can be produced using any one of a number of available circuit simulation programs.[5,6]

As V_{ds} is increased, not all of the drain to source voltage is available for reducing the transit time. Drain voltage in excess of one threshold below the gate voltage creates a short region of high electric field adjacent to the drain which the carriers cross very quickly. The electric field in the major portion of the channel from the source up to this region is proportional to $V_{gs} - V_{th}$, as shown in figure 1e. For $V_{ds} > (V_{gs} - V_{th})$, the drain current becomes independent of V_{ds}. Further increases in V_{ds} neither increase I_{ds} nor decrease the transit time. This range of V_{ds} values is known as saturation.

In saturation: $$I_{ds} = \frac{\mu\epsilon W(V_{gs} - V_{th})^2}{2LD} \qquad \text{[eq.5]}$$

With the exception of the factor of 2 in the denominator, this equation is similar to equation 3, with the V_{ds} factor in that equation replaced by its maximum effective value, $V_{gs} - V_{th}$. The factor of 2 in equation 5 arises from the non-uniformity of the electric field in the channel region when in saturation[1,R4].

The Basic Inverter

The first logic circuit we will describe is the basic digital inverter. Analysis of this circuit is then extended to analysis of basic NAND and NOR logic gates. The inverter's logic function is to produce an output which is the complement of its input. When describing the logic function of circuits in integrated systems, we assign the value logic-1 to voltages equaling or exceeding some defined logic threshold voltage, and logic-0 to voltages less than this threshold voltage.

Were there an efficient way to implement resistors in the MOS technology, we could build a basic digital inverter circuit using the configuration of figure 2a. Here, if the inverter input voltage V_{in} is less than the transistor threshold voltage $V_{th,}$ then the transistor is switched off, and V_{out} is "pulled-up" to the positive supply voltage VDD. In this case the output is the complement of the input. If V_{in} is greater than $V_{th,}$ the transistor is switched on and current flows from the VDD supply through the resistor R to GND. If R were sufficiently large, V_{out} could be "pulled-down" well below V_{th}, thus again complementing the input. However, the resistance per unit length of minimum width lines of various available conducting elements is far less than the effective resistance of the switched on MOSFET. Implementing a sufficiently large inverter pullup using resistive lines would require a very large area compared to that occupied by the transistor itself.

To circumvent this problem a *depletion mode* MOSFET is used as a pullup for the basic inverter circuit, symbolized and configured as shown in figure 2b. In contrast to the usual enhancement mode transistor, the depletion mode transistor has a threshold voltage, V_{dep}, that is less than zero. During fabrication, one of the masks is used to select any desired subset of transistors in the integrated system for processing as depletion mode transistors. For a depletion mode transistor to turn off, it requires a voltage on its gate relative to its source that is more negative than V_{dep}. But the depletion mode pullup transistor's gate is connected to its source, and thus it is always turned on. Hence, when the enhancement mode transistor is turned off, for example by connecting zero voltage to its gate, the output of the inverter will be equal to VDD. We will find that for reasonable ratios of the gate geometries of the two transistors, input voltages above a defined logic threshold voltage, V_{inv}, will produce output voltages below that logic threshold voltage, and vice versa.

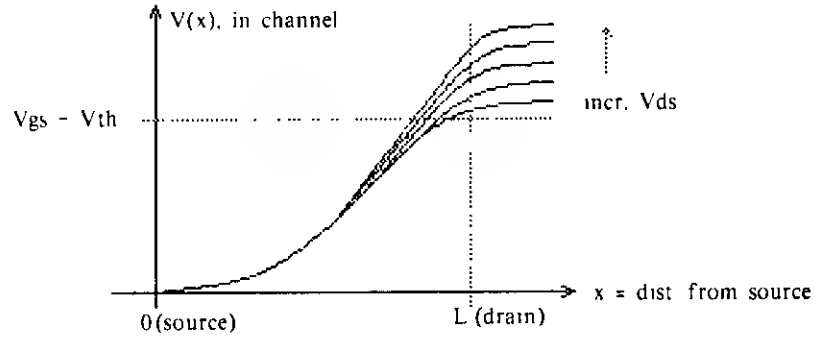


Fig. 1e. Voltage Profile Across Channel

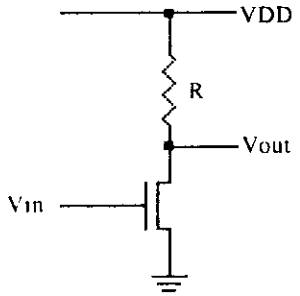


Fig. 2a. An Inverter

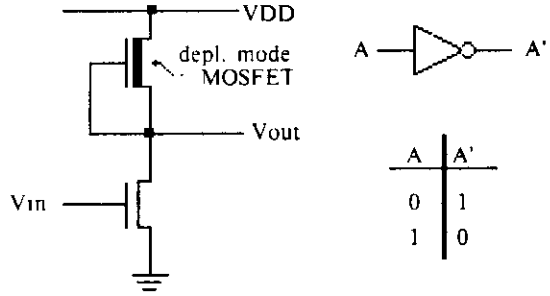


Fig. 2b. The Basic Inverter Circuit Diagram, Logic Symbol, Logic Function

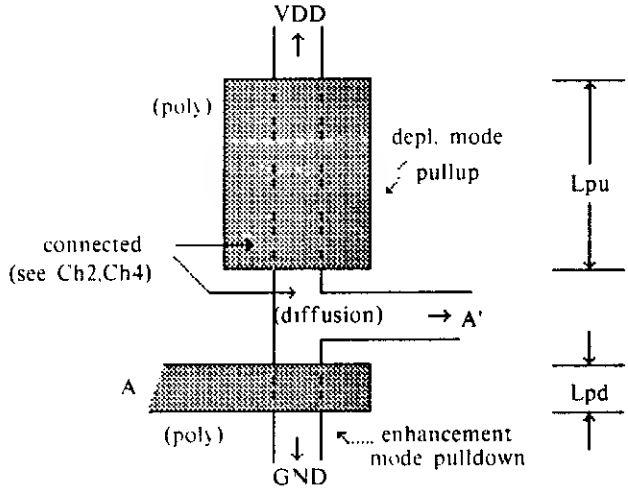


Fig. 2c. Basic Inverter Layout

The top view of the layout of an inverter on the silicon surface is sketched in figure 2c. It consists of two polysilicon regions overhanging a path in the diffusion level which runs between VDD and GND. This arrangement forms the two MOS transistors of the inverter. The inverter input A is connected to the poly forming the gate of the lower of the two transistors. The pullup is formed by connecting the gate of the upper transistor to its source. The fabrication details of such connections are described in chapter 2. The output of the inverter is shown emerging on the diffusion level, from between the drain of the pulldown and the source of the pullup. The pullup is a depletion mode transistor, and it is usually several times longer than the pulldown, to achieve the proper inverter logic threshold.

Figures 3a and 3b show the characteristics of a typical pair of MOS transistors used to implement an inverter. The relative locations of the saturation regions of the pullup and pulldown differ in these characteristics, due to the difference in their threshold voltages.

We can use a graphical construct to determine the actual transfer characteristic, V_{out} vs V_{in}, of the inverter circuit. From figure 2b we see that the V_{ds}(enh) of the enhancement mode transistor equals VDD minus V_{ds}(dep) of the depletion mode transistor. Also, V_{ds}(enh) equals V_{out}. In a steady state and with no current drawn from the output, the I_{ds} of the two transistors are equal. Since the pullup has its gate connected to its source, only one of its characteristic curves is relevant, namely the one for V_{gs}(dep) = 0. Taking these facts into account, we begin the graphical solution (fig. 3c) by superimposing plots of I_{ds}(enh) vs V_{ds}(enh), and the one plot of I_{ds}(dep) vs [VDD - V_{ds}(dep)]. Since the currents in both transistors must be equal, the intersections of these sets of curves yields V_{ds}(enh) = V_{out} versus V_{gs}(enh) = V_{in}. The resulting transfer characteristic is plotted in figure 3d.

Studying figures 3c and 3d, consider the effect of starting with V_{in} = 0 and then gradually increasing V_{in} towards VDD. While the input voltage is below the threshold of the pulldown transistor, no current flows in that transistor, the output voltage is constant at VDD, and the drain to source voltage across the pullup transistor is equal to zero. When V_{in} is first increased above the enhancement mode threshold, current begins to flow in the pulldown transistor. The output voltage decreases slowly as the input voltage is first increased above V_{th}. Subsequent increases in the input voltage rapidly lower the pulldown's drain to source voltage, until the point is reached where the pulldown leaves its saturation region and becomes resistive. Then as V_{in} continues to increase, the output voltage asymptotically approaches zero. The input voltage at which V_{in} = V_{out} is known as the logic threshold voltage V_{inv}. Figure 3d also shows the effect of changes in the transistor

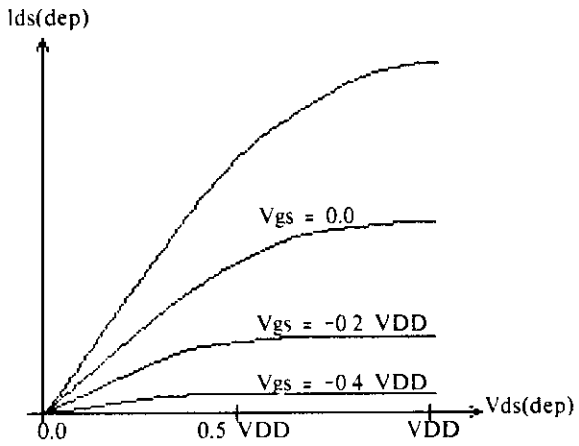


Fig. 3a. Inverter Pullup Characteristics

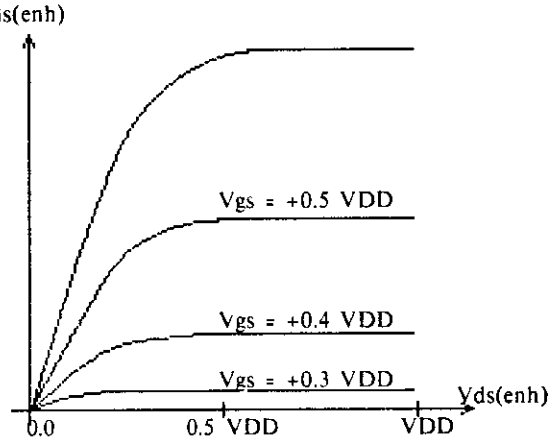


Fig. 3b. Inverter Pulldown Characteristics

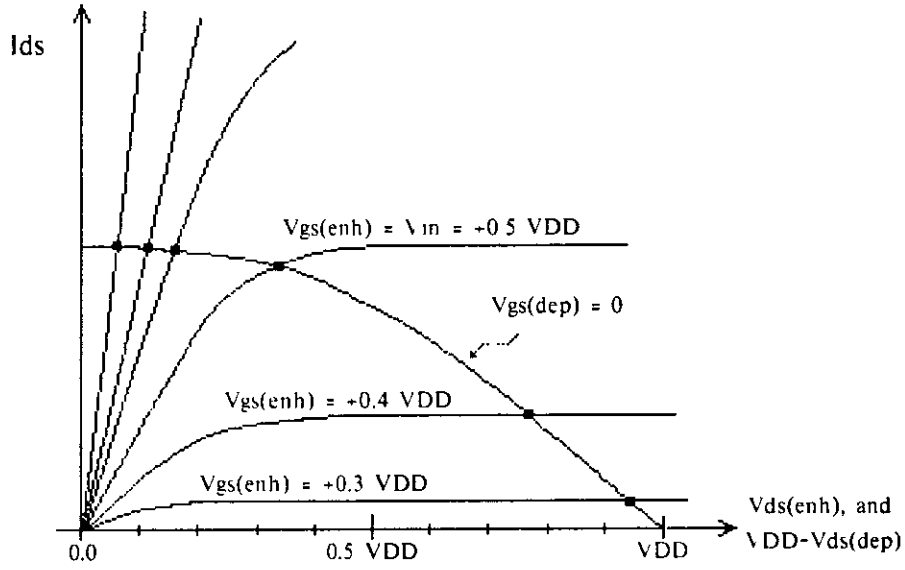


Fig. 3c. Ids(enh) vs Vds(enh), and Ids(dep) vs [VDD - Vds(dep)]

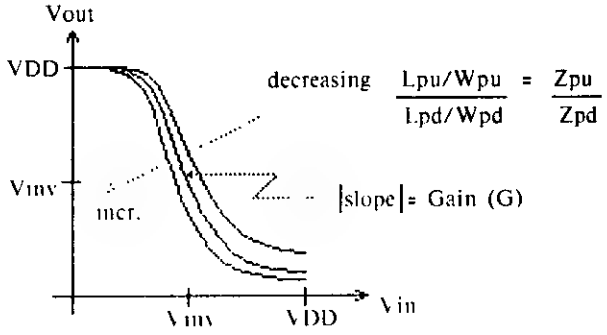


Fig. 3d. Vout vs Vin for the Basic Inverter

length to width ratios on the transfer characteristics and on the logic threshold voltage. The resistive impedance of the MOS transistor is proportional to the length to width ratio Z of its gate region. Using the subscript pu for the pullup transistor and pd for the pulldown transistor: If $Z_{pu} = L_{pu}/W_{pu}$ is increased relative to $Z_{pd} = L_{pd}/W_{pd}$, then V_{inv} decreases, and vice-versa. The gain, or negative slope of the transfer characteristic near V_{inv}, increases as Z_{pu}/Z_{pd} increases. The gain G must be substantially greater than unity for digital circuits to function properly.

Inverter Logic Threshold Voltage

The most fundamental property of the basic inverter circuit is its logic threshold voltage, V_{inv}. The logic threshold here is *not* the same as V_{th} of the enhancement mode transistor, but is that voltage on the input of the enhancement mode transistor which causes an equal output voltage. If V_{in} is increased above this logic threshold, V_{out} falls below it, and if V_{in} is decreased below V_{inv}, V_{out} rises above it. The following simple analysis assumes that both pullup and pulldown are in saturation, so that equation 3 applies. Usually the pullup is not quite in saturation, but the following is still nearly correct. V_{inv} is approximately that input voltage which would cause saturation current through the pulldown transistor to be equal to saturation current through the pullup transistor. Referring to eq.5, we find the condition for equality of the two currents given in eq.6.

Currents equal when:

$$\frac{Wpd}{Lpd}(V_{inv} - V_{th})^2 = \frac{Wpu}{Lpu}(-V_{dep})^2 , \qquad \text{[eq.6]}$$

or thus when:

$$V_{inv} = V_{th} - V_{dep}/[Zpu/Zpd]^{1/2} \qquad \text{[eq.6a]}$$

Here we note that the current through the depletion mode transistor is dependent only on its geometry and threshold voltage V_{dep}, since its $V_{gs} = 0$. Note that V_{inv} is dependent upon the thresholds of both the enhancement and depletion mode transistors, and also the square root of the ratio of the $Z = L/W$ of the enhancement mode transistor to that of the depletion mode transistor.

Tradeoffs are possible between these threshold voltages and the areas and current driving capability of transistors in the system's inverters. To maximize (V_{gs} - V_{th}) and increase the pulldowns' current driving capability for a given area, V_{th} should be as low as possible. However, if V_{th} is too low, inverter outputs won't be driveable below V_{th}, and inverters won't be able to turn off transistors used as simple switches. The original choice of V_{th} ~ 0.2VDD is a reasonable compromise here.

Similarly, to maximize the current driving capability of pullups of given area, we might set the system's V_{dep} as far negative as possible. However, eq. 4a shows that for chosen V_{inv} and V_{th}, decreasing V_{dep} requires an increase in L_{pu}/W_{pu}, typically leading to an increase in pullup area. The compromise made in this case is usually as follows. The negative threshold of depletion mode transistors is set during fabrication such that with gate tied to source, they turn on approximately as strongly as would an enhancement mode transistor with VDD connected to its gate and its source grounded. In other words, depletion mode and enhancement mode transistors of equal gate dimensions would have equal drain to source currents under those conditions. Applying eq.5 in those conditions we find that:

$$(-V_{dep})^2 \sim (VDD - V_{th})^2.$$

Therefore, $-V_{dep} \sim (VDD - V_{th})$, and $V_{dep} \sim -0.8VDD$. While adjustments in the details of this choice are often made in the interest of optimization of processes for a particular product, we will assume here this approximate equality of turn-on voltages of the two transistor types for the sake of simplicity. Substituting this choice of V_{dep} into eq.6a, we find that for V_{th} small compared to VDD:

$$V_{inv} \sim VDD/[Zpu/Zpd]^{1/2} \qquad \text{[eq.7]}$$

In general it is desirable that the margins around the inverter threshold be approximately equal, i.e., that the inverter threshold, V_{inv}, lie approximately midway between VDD and ground. We see from eq.7 that this criterion is met by a ratio of pullup Z to pulldown Z of approximately 4:1. We will see later that the choice of $V_{dep} \sim VDD - V_{th}$, producing a ratio of 4:1 here, will lead to a balancing of performances in certain other important circuits.

Inverter Delay

A minimum requirement for an inverter is that it drive another identical to itself. Let us analyze the delay through a string of inverters of identical dimensions since this is the simplest case in which we can estimate performance. Inverters connected in this way are shown in Fig. 4a. We define the the inverter ratio k as the ratio of Z of the pullups to Z of the pulldowns. Note that we will sometimes use the alternative pullup symbol shown ("resistor with gate"), to clarify its functional purpose.

Let us assume that prior to t = 0, the voltage at the input of the first inverter is zero, and hence, the voltage output of the second inverter will be low. At time t=0, let us place a voltage equal to VDD on the input of the first inverter and follow the sequence of events which follows. The output of the first inverter, which leads to the gate of the second inverter, will initially be at VDD. Within approximately one transit time, the pulldown transistor of the first inverter will remove from this node an amount of charge equal to VDD times the gate capacitance of the pulldown of the second inverter. The pullup transistor of the second inverter is now faced with the task of supplying a similar charge to the gate of the third inverter, to raise it to VDD. Since it can supply at most only 1/k'th of the current that can be supplied by the pulldown transistor, the delay in the second inverter stage is approximately k times that of the first.

It is thus convenient to speak of the *inverter pair delay* which includes the delay for one lowgoing transition and one highgoing transition. This inverter pair delay is approximately (1+k) times the transit time, as shown in figure 4a. The fact that the rising transition is slower than the falling transition by approximately the inverter transistors' geometry ratios is an inherent characteristic of any ratio type logic. It is not true of all logic families. For example, in families such as complementary MOS where there are both pMOS and nMOS devices on the same silicon chip and both types operate strictly as pulldown enhancement mode devices, any delay asymmetry is a function of the difference in mobilities of the p and n type charge carriers rather than of the transistor geometrical ratios.

Fig. 4b shows an inverter driving the inputs of several other inverters. In this case, for a fanout factor f, it is clear that in either the pullup or pulldown direction, the active device must supply f times as much charge as it did in the case of driving a single input. In this case, the delay both in the up and downgoing directions is increased by approximately the factor f. In the case of the downgoing transition, the delay is approximately f times the

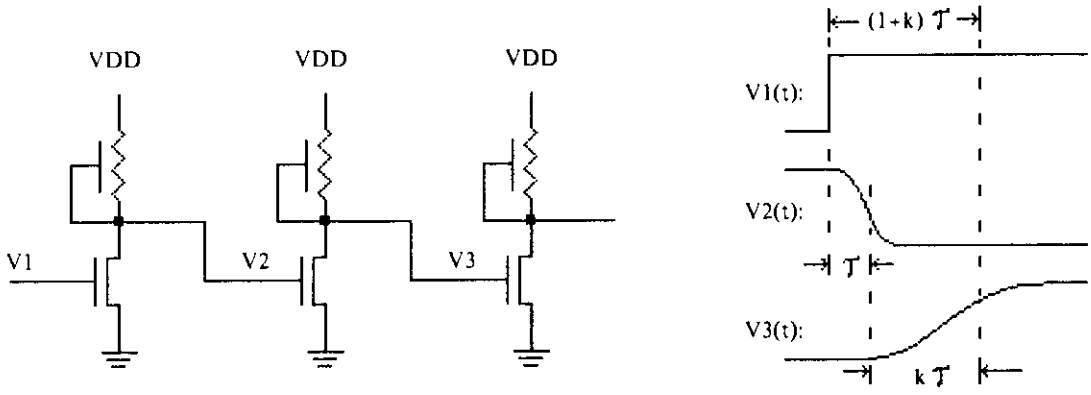
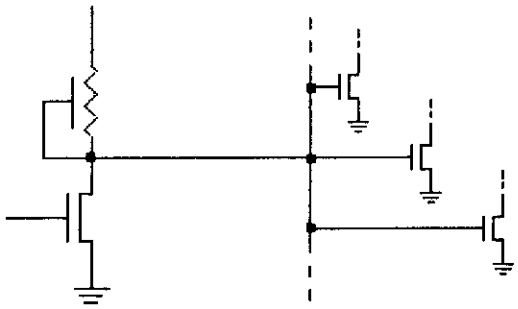


Fig. 4a. Inverter Delay

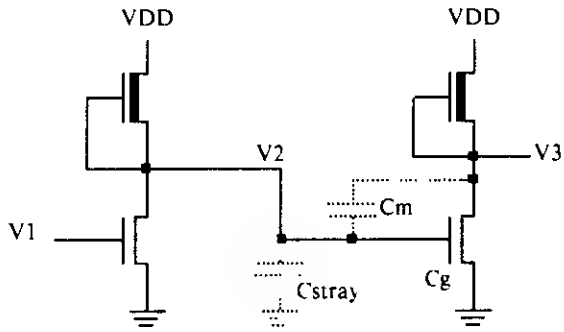


For fanout of f:

down delay ~ f T

up delay ~ kf T

Fig. 4b. Fanout



Start:

V2 = 0, V3 = VDD

Qg = 0, Qm = -CmVdd

Qwire = 0

Finish:

V2 = VDD, V3 = 0

Qg = CgVDD, Qm = +CmVDD

Qwire = CstrayVDD

Fig. 4c. The Miller Effect

[total effective input capacitance = $\frac{dQ}{dV} = \frac{(Qfinish - Qstart)}{(Vfinish - Vstart)} = Cg + 2Cm + Cstray$]

transit time of the pulldown transistor, and in the case of the upgoing transition, the delay is approximately the inverter ratio k times the fanout factor times the pulldown transit time.

In the discussions of transit time given earlier, it was assumed that both the depletion mode pullup device and the enhancement mode pulldown device were operating in the resistive region. It was also assumed that all capacitances were constant, and not a function of voltage. These conditions are not strictly met in the technology we are discussing. Delay calculations given in this text are based on a "switching model" where individual stages spend a small fraction of their time in the mid-range of voltages around V_{inv}. This assumption introduces a small error of the order of 1/G. Because of these and other second order effects, the switching times actually observed vary somewhat from those derived.

Parasitic Effects

In integrated systems, capacitances of circuit nodes are due not only to the capacitance of gates connected to the nodes, but also include capacitances to ground of signal paths connected to the nodes and other stray capacitances. These other capacitances, sometimes called parasitic or stray capacitances, are not negligible. While gate capacitances are typically an order of magnitude greater per unit area than the capacitances of the signal paths, the signal paths are often much larger in area than the associated gate regions. Therefore, a substantial fraction of the delay encountered may be accounted for by stray capacitance rather than by the inherent properties of the active transistors. In the simplest case where the capacitance of a node is increased by the presence of parasitic area attached to the node, the delays can be accounted for by simply increasing the transit time by the ratio of the total capacitance to that of the gate of the transistor being driven. Time is required to supply charge not only to the gate itself but also to the parasitic capacitance.

There is one type of parasitic, however, which is not accounted for so simply. All MOS transistors have a parasitic capacitance between the drain edge of the gate and the drain node. This effect is shown schematically in figure 4c. In an inverter string, this capacitance will be charged in one direction for one polarity of input, and in the opposite direction for the opposite polarity input. Thus, on a gross scale its effect on the system is twice that of an equivalent parasitic capacitance to ground. Therefore, gate to drain capacitances should be approximately doubled, and added to the gate capacitance C_g and the stray capacitances, to account for the total capacitance of the node and thus for the effective delay time of the inverter. The effective inverter pair delay then = $\tau(1+k)C_{total}/C_g$.

Driving Large Capacitive Loads

As we have seen, the delay per inverter stage is multiplied by a fanout factor. The overall performance of a system may be seriously degraded if it contains any large fanouts, where one circuit within the system is required to drive a large capacitive load. As we shall see, this situation often occurs in the case of control drivers which are required to drive a large number of inputs to memory cells or logic function blocks. A similar and even more serious problem is driving wires which go off the silicon chip to other chips or input/output devices. In such cases the ratio of the capacitance which must be driven to the inherent capacitance of a gate circuit on the chip is often many orders of magnitude, causing a serious delay and a degradation of system performance.

Consider how we may drive a capacitive load C_L in the minimum possible time given that we are starting with a signal on the gate of an MOS transistor of capacitance C_g. Define the ratio of the load capacitance to the gate capacitance, C_L/C_g, as Y. It seems intuitively clear that the optimum way to drive a large capacitance is to use our elementary inverter to drive a larger inverter and that larger inverter to drive a still larger inverter until at some point the larger inverter is able to drive the load capacitance directly. Using an argument similar to the fanout argument it is clear that for one inverter to drive another inverter, where the second is larger in size by a factor of f, results in a delay f times the inherent inverter delay, τ. If N such stages are used, each larger than the previous by a factor f, then the total delay of the inverter chain is $Nf\tau$, where f^N equals Y. Note that if we use a large factor f, we can get by with few stages, but each stage will have a long delay. If we use a smaller factor f, we can shorten the delay of each stage, but are required to use more stages. What value of N minimizes the overall delay for a given Y? We compute this value as follows:

Since $f^N = Y$, $\ln(Y) = N \ln(f)$

Delay of one stage $= f\tau$,

Thus total delay is $= Nf\tau = \ln(Y)[\, f/\ln(f)\,]\tau$ [eq.8]

Notice that the delay is always proportional to ln(Y), a result of the exponential growth in successive stages of the driver. The multiplicative factor, $f/\ln(f)$, is plotted as a function of

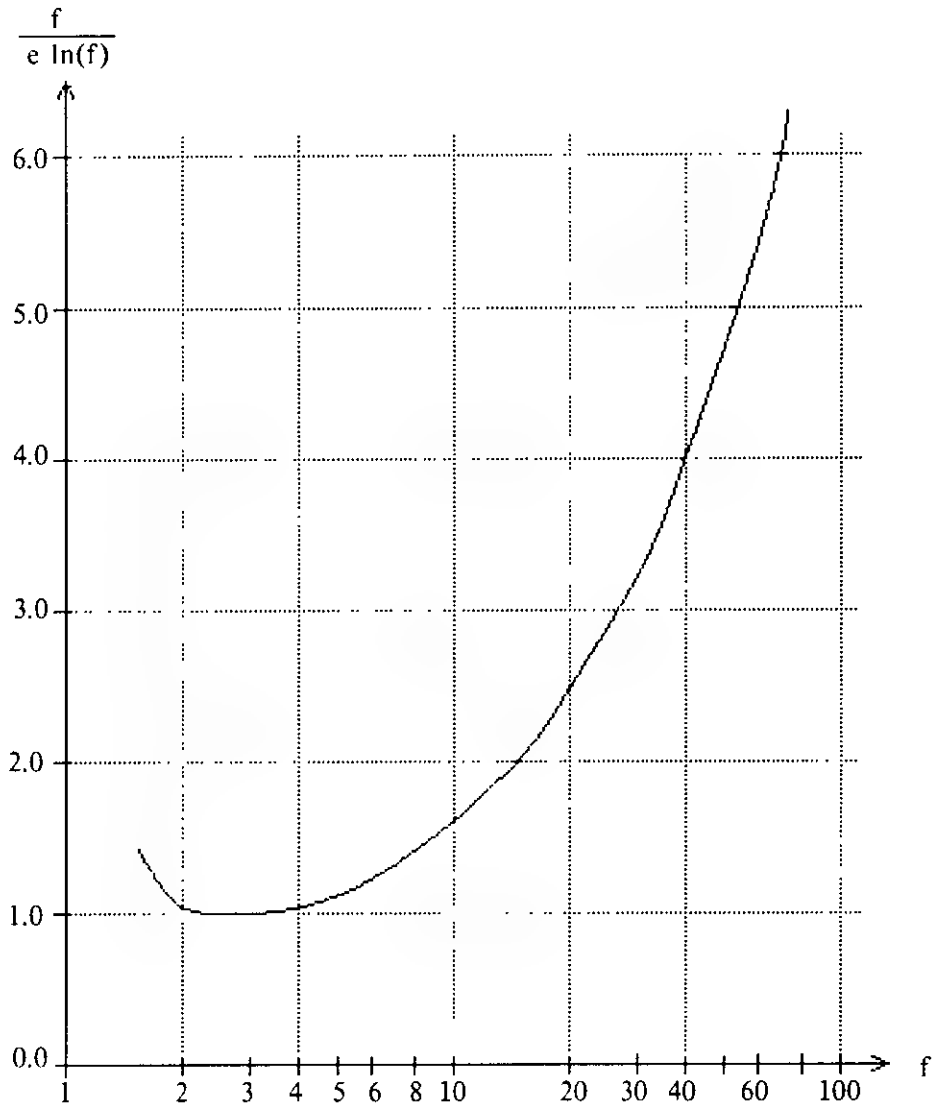


Fig. 5. Relative Time Penalty, $\frac{f}{e \ln(f)}$, vs Size Factor f

f in figure 5, normalized to its minimum value (e). Total delay is minimized when each stage is larger than the previous one by a factor of e, the base of natural logarithms. Minimum total delay is the elementary inverter delay τ times e times the natural logarithm of the ratio of the load capacitance to the elementary inverter capacitance.

$$\text{Min. total delay} \sim \tau e[\ln(C_L/C_g)] \qquad \text{[eq.9]}$$

Minimum delay through the driver is seldom the only design criterion. The relative time penalty introduced by the choice of other values of f can be read directly from figure 5.

Space vs Time

From the results of the sections on inverter delay, parasitic effects, and driving large capacitances, we see that areas and distances on the silicon surface trade off against delay times. For an inverter to drive another inverter some distance away, it must charge not only the gate capacitance of the succeeding inverter but also the capacitance to ground of the signal path connecting the two. Increasing the distance between the two inverters will therefore increase the inverter pair delay. This effect can be counterbalanced by increasing the area of the first inverter, so as to reduce the ratio of the load capacitance to the gate capacitance of the first inverter. But the delay of some previous driving stage is then increased. There is no way to get around the fact that transporting a signal from one node to another some distance away requires charging or discharging capacitance, and therefore takes time. Note that this is not a velocity of light limitation as is often the case outside the chip. The times are typically several orders of magnitude longer than those required for light to traverse the distances involved. To minimize both the time and space required to implement system functions, we will tend to use the smallest possible circuits and locate them in ways which tend to minimize the interconnection distances.

The results of a previous section can be used here to illustrate another interesting space vs time effect. Suppose that the minimum size transistors of an integrated system have a transit time τ and gate capacitance C_g. A minimum size transistor within the system produces a signal which is then passed through successively larger inverting logic stages and eventually drives a large capacitance C_L with minimum total delay equal to t_{min}. With the passage of time, fabrication technology improves and we replace the system with one in which the circuits are all scaled down in size by dividing by a factor α (the motivation for

this is clear: the new system may contain α^2 as many circuits). As described in a later section, we will find that the transit times of the smallest circuits will now be $\tau' = \tau/\alpha$, and their gate capacitance will be $C_g' = C_g/\alpha$. Referring to equation 5a., we find that the new minimum total delay, t_{min}', to drive C_L scales less favorably:

$$t_{min}' = t_{min}[(1/\alpha)\ln(\alpha)]$$

Therefore, as the inverters scale down and τ gets smaller, more inverting logic stages are required to obtain the minimum offchip delay. Thus the relative delay to the outside world becomes larger. However, the absolute delay becomes smaller.

Basic NAND and NOR Logic Circuits

NAND and NOR logic circuits may be constructed in integrated systems as simple expansions of the basic inverter circuit. The analysis of the behavior of these circuits, including their logic threshold voltages, transistor geometry ratios and time delays, is also a direct extension of the analysis of the basic inverter.

The circuit layout diagram of a two input NAND gate is shown in figure 6a. The layout is that of a basic inverter with an additional enhancement mode transistor in series with the pulldown transistor. NAND gates with more inputs may be constructed by simply adding more transistors in series with the pulldown path. The electrical circuit diagram, truth table and logic symbol for the two input NAND gate are shown in figure 6b. If either of the inputs A or B is a logic-0, the pulldown path is open and the output will be high, and therefore a logic-1. For the output to be driven low, to logic-0, both inputs must be high, at logic-1. The logic threshold voltage of this NAND gate is calculated in a similar manner to that of the basic inverter, except equation 7 is rewritten with the length of the pulldowns replaced with the sum of the lengths of the two pulldowns (assuming their widths are equal) as follows:

$$V_{thNAND} \sim VDD/[(L_{pu}/W_{pu})/((L_{pd_a}+L_{pd_b})/W_{pd})]^{1/2}$$

This equation indicates that as pulldowns are added in series to form NAND gate inputs, the pullup length must be enlarged to hold the logic threshold voltage constant.

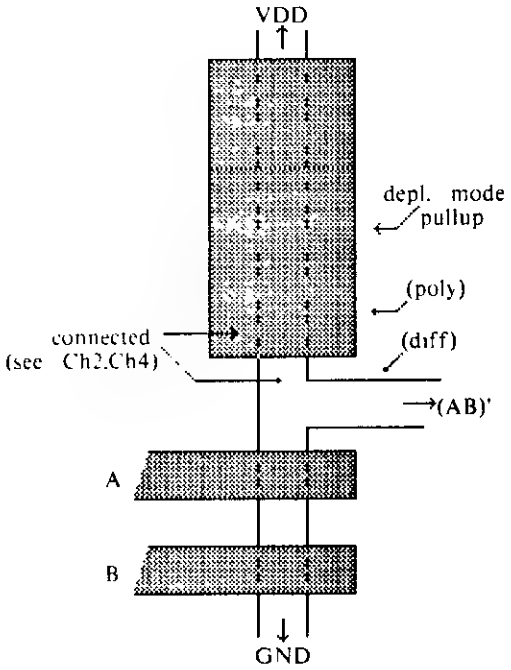
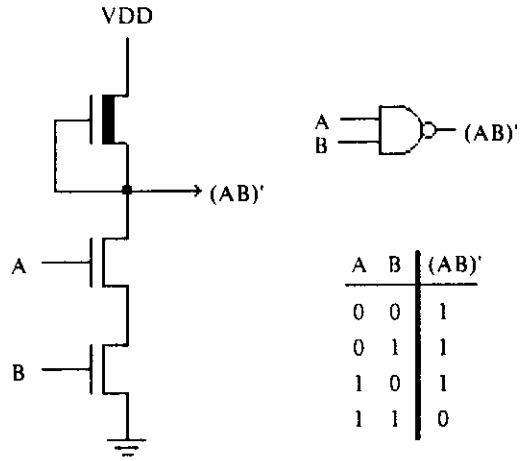


Fig. 6a. NAND Gate
[top view of layout]



A	B	(AB)'
0	0	1
0	1	1
1	0	1
1	1	0

Fig. 6b. NAND Gate Circuit Diagram, Logic Symbol, Logic Function

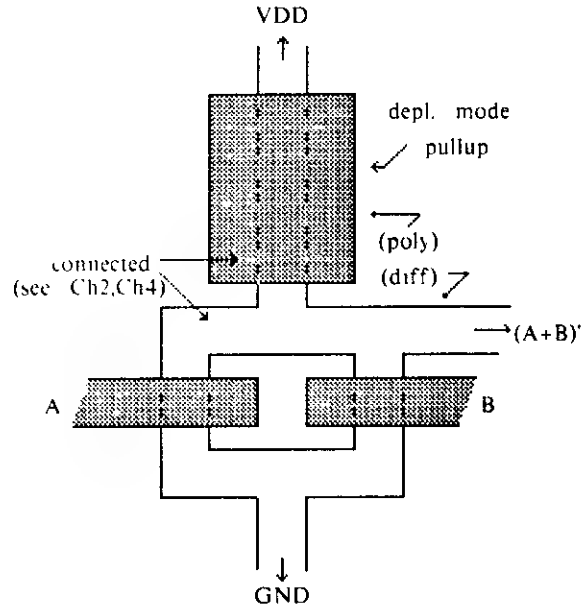
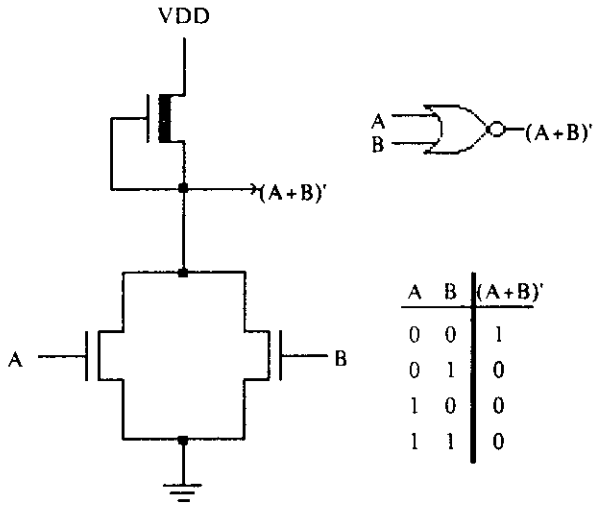


Fig. 6c. NOR Gate
[top view of layout]



A	B	(A+B)'
0	0	1
0	1	0
1	0	0
1	1	0

Fig. 6d. NOR Gate Circuit Diagram, Logic Symbol, Logic Function

The logic threshold voltage of an n-input NAND gate, assuming all the pulldowns have equal geometries, is:

$$V_{thNAND} \sim VDD/[(Lpu/Wpu)/(nLpd/Wpd)]^{1/2}$$

As inputs are added and pullup length is increased, the delay time of the NAND gate is also correspondingly increased, both for rising and falling transitions.

$$\tau_{NAND} \sim n\tau_{inv}$$

The circuit layout diagram of a two input NOR gate is shown in figure 6c. The layout is that of a basic inverter with an additional enhancement mode transistor in parallel with the pulldown transistor. Additional inputs may be constructed by simply placing more transistors in parallel with the pulldown path. The circuit diagram, truth table and logic symbol for the two input NOR gate are shown in figure 6d. If either of the inputs A or B is a logic-1, the pulldown path to ground is closed and the output will be low, and therefore a logic-0. For the output to be driven high, to logic-1, both inputs must be low, at logic-0. If one of its inputs is kept at logic-0, and the other swings between logic-0 and logic-1, the logic threshold voltage of the NOR gate is the same as that of a basic inverter of equal pullup to pulldown ratio. If this ratio were 4:1 to provide equal margins, then $V_{thNOR} \sim$ VDD/2 with only one input active. However, if both pulldowns had equal geometries, and if both inputs were to move together between logic-0 and logic-1, V_{thNOR} would be reduced to $\sim VDD/(8)^{1/2}$. The logic threshold voltage of an n-input NOR circuit decreases as a function of the number of active inputs (inputs moving together from logic-0 to logic-1). The delay time of the NOR gate with one input active is the same as that of an inverter of equal transistor geometries, except for added stray capacitance. Its delay time for falling transitions is decreased as more of its inputs are active.

Super Buffers

As we have noted, ratio type logic suffers from an asymmetry in its ability to drive capacitive loads. This asymmetry results from the fact that the pullup transistor has of necessity less driving capability than the pulldown transistor. There are, however, methods for avoiding this asymmetry. Shown in figures 7a and 7b are circuits for inverting and non-inverting drivers which are approximately symmetrical in their capability of sourcing or sinking charge into a capacitive load. Drivers of this type are called *super buffers.*

Both types of super buffer are built using a depletion mode pullup transistor and an enhancement mode pulldown transistor, with a ratio of Z's of approximately 4:1 as in the basic inverter. However, the gate of the pullup transistor, rather than being tied to its source, is tied to a signal which is the complement of that driving the pulldown transistor. When the pulldown transistor gate is at a high voltage, the pullup transistor gate will be approximately at ground, and the current through the super buffer will be similar to that through a standard inverter of the same size. However, when the gate of the pulldown transistor is put to zero, the gate of the pullup transistor will go rapidly to VDD since it is the only load on the output of the previous inverter, and the depletion mode transistor will be turned on at approximately twice the drive which it would experience if its gate were tied to its source. Since the current from a device in saturation goes approximately as the square of the gate voltage, the current sourcing capability of a super buffer is approximately four times that of a standard inverter. Hence, the current sourcing capability of its pullups is approximately equal to the current sinking capability of its pulldowns, and wave forms from super buffers driving capacitive loads are nearly symmetrical.

The effective delay time, τ, of super buffers is thus reduced to approximately the same value for highgoing and lowgoing wave forms. Needless to say, when large capacitive loads are to be driven, super buffers are universally used. The arguments used in the last section to determine how many stages are used to drive a large capacitive load from a small source apply directly to super buffers. For that reason we have not explicitly indicated an inverter ratio k in that section

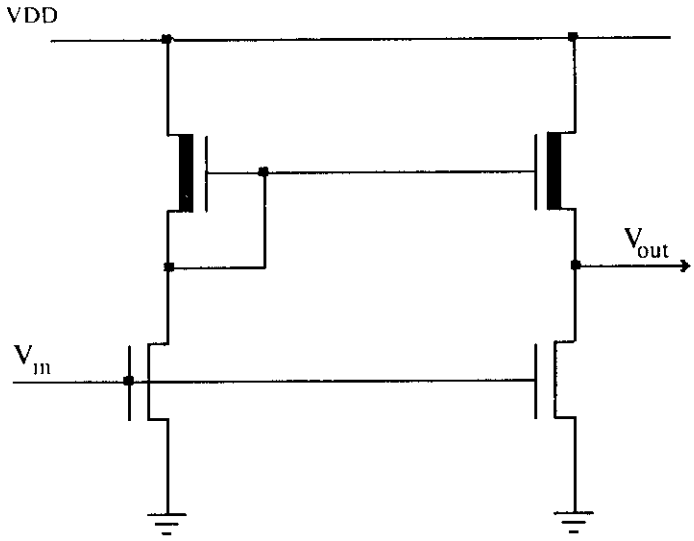


Fig. 7a. Inverting Super Buffer

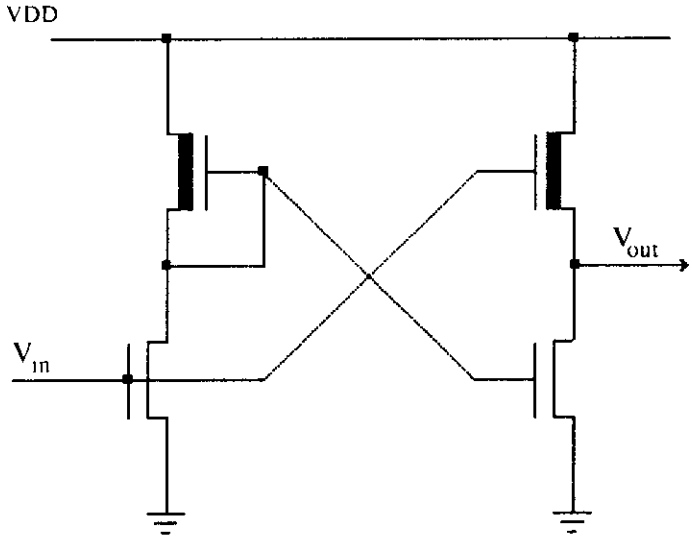


Fig. 7b. Non-Inverting Super Buffer

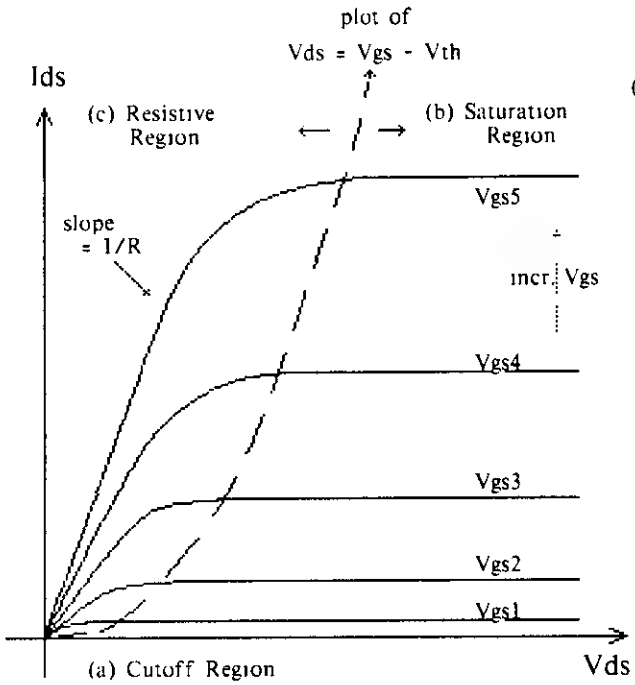
A Closer Look at the Electrical Parameters

Up to this point we have talked in very simple terms about the properties of the MOS transistors. They have a capacitance associated with their gate input and a transit time for electrons to move from the source to the drain. We have given simple expressions for the drain to source current. For very low V_{ds}, the MOS transistor's drain to source path acts as a resistor whose conductance is directly proportional to the gate voltage above threshold, as given in equation 3. For V_{ds} larger than $V_{gs} - V_{th}$, the device acts as a current source, with a current proportional to $(V_{gs} - V_{th})^2$, as given in equation 5. As V_{ds} passes through the intermediate range between these two extremes, there is a smooth transistion between the two types of behavior, as given in equation 10.[1]

$$I_{ds} = Q/\tau = \mu C_g[(V_{gs} - V_{th})\, V_{ds} - (V_{ds}^2)/2]/L^2 \qquad \text{[eq.10]}$$

Figure 9a plots I_{ds} vs V_{ds}, summarizing the various regions of MOS transistor operation.

There is another electrical characteristic we may occasionally have to take into account. The threshold voltage of an MOS transistor is not a constant, but varies slightly as a function of the voltage between the source terminal of the transistor and the silicon substrate, which is usually at ground. This so called *body effect* is illustrated in figure 9b. If the source to bulk (substrate) voltage, V_{sb} , equals zero, then V_{th} is at its minimum value of approximately 0.2 VDD. As V_{sb} is increased, V_{th} increases slightly. For typical processes, V_{th} reaches a maximum value of about 0.3 VDD for $V_{sb} \sim$ VDD. V_{dep} is similarly affected, ranging from about -0.8 VDD to -0.7 VDD as the V_{sb} of a depletion mode MOSFET is raised from zero to VDD volts. As shown in figure 9b, it is possible to insert a fixed bias voltage between the circuit ground and the substrate (rather than just connect them). Such a *substrate bias* provides an electrical mechanism for setting the threshold to an appropriate value.



(a) Cutoff Region:

Vgs < Vth , Ids = 0

(b) Saturation Region:

Vgs ≥ Vth , Vds sufficiently high so Vgd < Vth . (i.e., Vds ≥ Vgs-Vth)

MOSFET acts as current source, with Ids proportional to $(Vgs - Vth)^2$

(c) Resistive Region:

Vgs ≥ Vth , Vds sufficiently low so Vgd ≥ Vth . (i.e.: Vds < Vgs - Vth)

MOSFET acts as resistor, with resistance inversely proportional to (Vgs - Vth)

Fig. 9a. Summary of MOS Transistor Characteristics

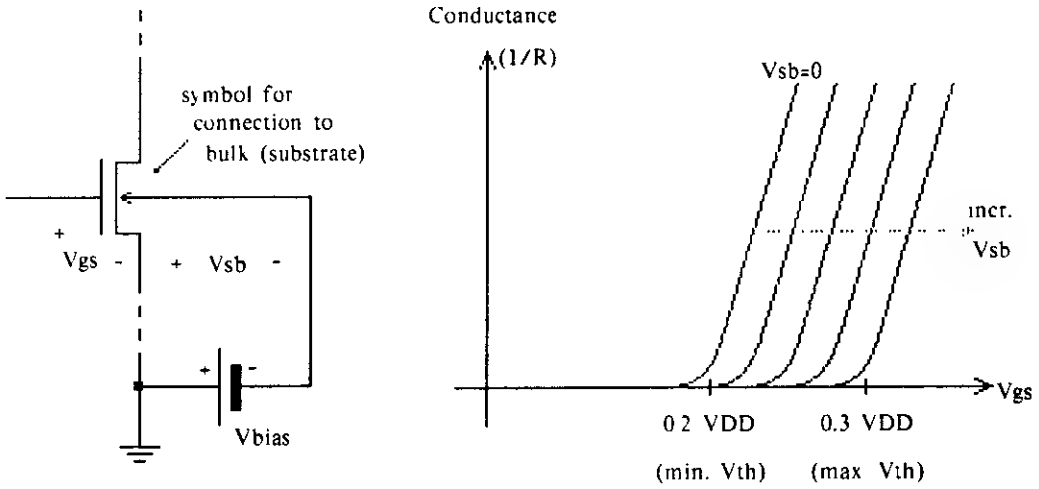


Fig. 9b. The Body Effect

Depletion Mode vs Enhancement Mode Pullups

With its gate tied to VDD, an enhancement mode transistor will be on for all $V_{ds} > V_{th}$, and thus can be used for a pullup device in inverting logic circuits. Early MOS processes used pullup devices of exactly this type.

In this section we will make a comparison of the rising transients of the two types of pullup circuits. As noted earlier, rising transients in ratio type logic are usually slower than falling transients, and thus rising transients generally have greater impact on system performance. In the simplest cases, this asymmetry in the transients results from the current sourcing capability of the pullup transistor being less than that of its pulldown counterpart. The simple intuitive time arguments given earlier are quite adequate for making estimates of system performance in most cases. However, there are situations in which the transient time may be much longer than a naive estimate would indicate. The rising transient of the enhancement mode pullup is one of these.

A depletion mode pullup transistor feeding a capacitive load is shown schematically in figure 10a. Since $V_{gs} \geq V_{th}$ and $V_{gd} \geq V_{th}$, the pullup transistor is in the resistive region. The final stages of the rising transient are given by the following exponential:

$$V(t) = VDD[1 - e^{-t/(RC_L)}]$$

Where for inverter ratio k, and pulldown transit time τ and gate capacitance C_g, the time-constant of the rising transient is given by:

$$RC_L = k\tau C_L/C_g$$

A somewhat more complicated situation is presented by an enhancement mode transistor sourcing charge into a capacitive load. This situation is shown schematically in Fig. 10b. Note that since $V_{gd} = 0$, the transistor is in saturation whenever $V_{gs} > V_{th}$. The problem with sourcing charge from the enhancement mode transistor is that as the voltage at the output node gets closer and closer to one threshold below VDD, the amount of current provided by the enhancement mode transistor decreases rapidly.

The dependence of the enhancement mode pullup current upon the output voltage V is given in eq. 11.

$$Q = - \frac{\varepsilon WL}{D} [(VDD - V_{th}) - V]$$

$$\tau = 2L^2/\mu[(VDD - V_{th}) - V]$$

$$I_{ds} = - Q/\tau = \frac{\mu\varepsilon W}{2LD}[(VDD - V_{th}) - V]^2 \qquad \text{[eq.11]}$$

The fact that the pullup current decreases as the output voltage nears its maximum value causes the rising transient from such a circuit to be of qualitatively different form than that of a depletion mode pullup. Equating $I_{ds} = C_L dV/dt$ with the expression in equation 11, and then solving for V(t), we find the rising voltage transient, for large t:

$$V(t) = VDD - V_{th'} - C_L \frac{LD}{\mu\varepsilon Wt} \qquad \text{[eq.12]}$$

Note that in this configuration, the threshold voltage Vth' of the pullup is near its maximum value as V(t) rises towards VDD, due to the body effect. A comparison of the rising transients of the preceding two circuits, assuming the same load capacitance and the same pullup source current at zero output voltage, is shown in Fig. 10c. The rising transient for the depletion mode pullup transistor is crisp and converges rapidly towards VDD. However, the rising transient for the enhancement mode pullup transistor, while starting rapidly, lags far behind and within the expected time response of the system, never even comes close to one threshold below VDD. Even for very large t, $V(t) < VDD - V_{th'}$. The practical effect of this property of enhancement mode transistors is that circuits designed to work from a voltage derived from the output of such a circuit should be designed with an inverter threshold V_{inv} considerably lower than that of circuits designed to work with the output of a depletion mode pullup circuit. In order to obtain equal inverter margins without sacrificing performance, we will normally use depletion mode pullups.

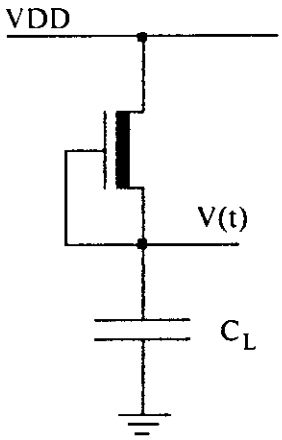


Fig. 10a. Depletion Mode MOSFET Pulling Up Capacitive Load

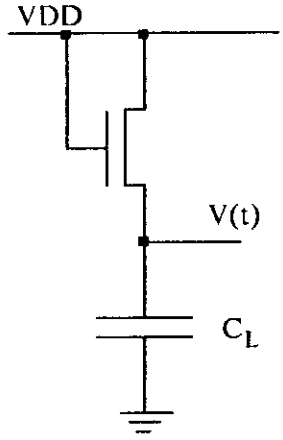


Fig. 10b. Enhancement Mode MOSFET Pulling Up Capacitive Load

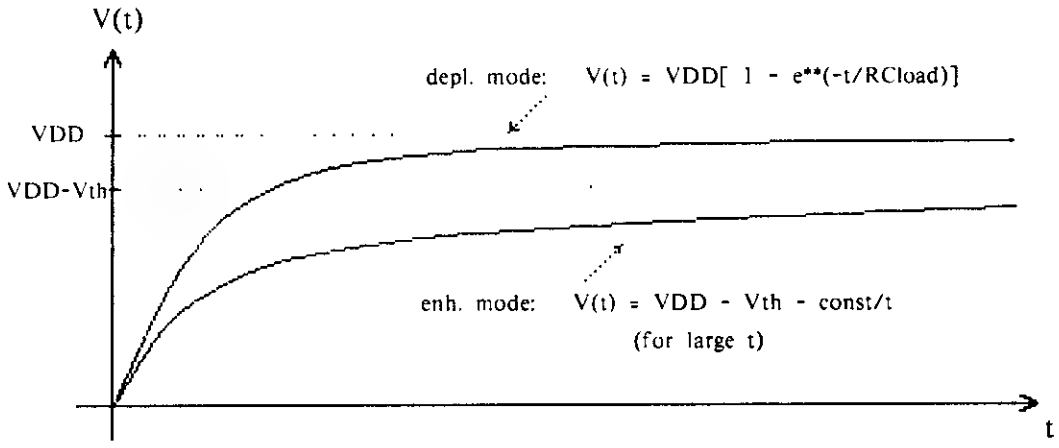


Fig. 10c. Comparisons of Rising Transients for the Two Types of Pullups

Delays in Another Form of Logic Circuitry

Enhancement mode transistors, when used in small numbers and driving small capacitive loads, may often be used as switches in circuits of simple topology to provide logic signal steering functions of much greater complexity than could be easily achieved in ratio type inverting logic. These circuits are reminiscent of relay switching logic, and transistors used in this way are referred to as "pass transistors" or "transmission gates". Example circuits using this type of design are given in Chapter 3. A particularly interesting example is the Manchester carry chain[4a,b], used for propagating carry signals in parallel adders. In each stage of the adder a carry propagate signal is derived from the two input variables to the adder, and if it is desired to propagate the carry, this propagate signal is applied to the gate of an enhancement mode pass transistor. The source of the transistor is carry-in to the present stage, and the drain of the transistor is carry-out to the next stage. In this way, a carry can be propagated from less to more significant stages of the adder without inserting a full inverter delay between stages. The circuit is shown schematically in Fig. 11a.

The delay through such a circuit does not involve inverter delays but is of an entirely different sort. A voltage along the chain divides into V_{ds} across each pass transistor. Thus V_{ds} is usually low, and the pass transistors operate primarily in the resistive region. We can think of each transistor as a series resistance in the carry path, and a capacitance to ground formed by the gate to channel capacitance of each transistor, and the strays associated with the source, drain, and connections with the following stage. An abstraction of the electrical representation is shown in Fig. 11b. The minimum value of R is the turned on resistance of each enhancement mode pass transistor, while the minimum value of C is the capacitance from gate to channel of the pass transistor. Strays will increase both values, especially that of C. The response at the node labelled V_2 with respect to time is given in eq. 13. In the limit as the number of sections in the network becomes large, eq. 13 reduces to the differential form shown in eq. 14 where R and C are now the resistance and capacitance per unit length, respectively.

$$C\, dV_2/dt = [(V_1 - V_2) - (V_2 - V_3]/R \qquad \text{[eq.13]}$$

$$RC\, dV/dt = d^2V/dx^2 \qquad \text{[eq.14]}$$

Equation 14 is the well-known diffusion equation, and while its solutions are complex, in general the time required for a transient to propagate a distance x in such a system is proportional to x^2. One can see qualitatively that this might be so. Doubling the number of sections in such a network doubles both the resistance and the capacitance, and therefore causes the time required for the system to respond to increase by a factor of approximately four. The response of a system of n stages to a step function input is shown in Fig. 11c.

If we add one more pass transistor to such a chain of n pass transistors, the added delay through the chain is small for small n, but very large for large n. Therefore, it is highly desirable to group the pass transistors used for steering, multiplexing, and carry-chain type logic into short sections and interpose inverting logic between these sections. This approach applied to the carry chain is shown in figure 11d. The delay through a section of n pass transistors is proportional to RCn^2. Thus the total delay ~ RCn^2 plus the delay through the inverter τ_{inv}. The average delay per stage is given in eq. 15. To minimize the delay per stage, chose n such that the delay through n pass transistors equals the inverter delay:

$$\text{Total delay} \sim RCn^2 + \tau_{inv},$$

$$\text{Average delay/stage} \sim RCn + \tau_{inv}/n \qquad \text{[eq.15]}$$

$$\text{Min. delay when:} \quad RCn^2 \sim \tau_{inv}$$

Since logic done by steering signals with pass transistors does not require static power dissipation, a generalization of this result may be formulated. It pays to put as much logic into steering type circuits as possible until there are enough pass transistors to delay the signal by approximately one inverting logic delay. At this point, the level of the signal can be restored by an inverting logic stage.

The pass transistor has another important advantage over an inverting logic stage. When used to control or steer a logic signal, the pass transistor has only an input, control, and output connections. A NAND or NOR logic gate implementing the same function, in addition to containing two more transistors and thus occupying more area, also requires VDD and GND connections. As a result, the topology of interconnection of pass transistor circuits is far simpler than that of inverting logic circuits. This topological simplicity of pass transistor control gates is an important factor in the system design concepts developed in later chapters.

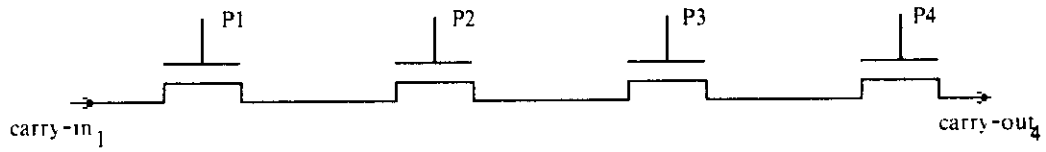


Fig. 11a. Pass Transistor Chain Propagating a Carry Signal

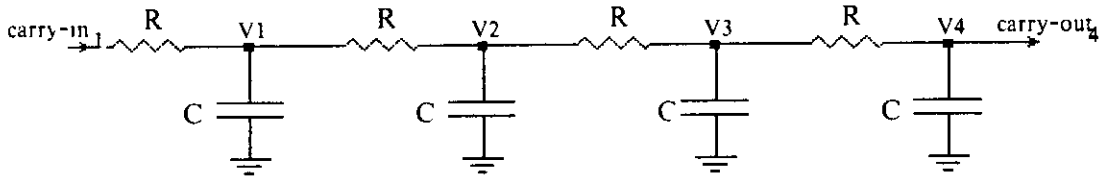


Fig. 11b. Equivalent Circuit

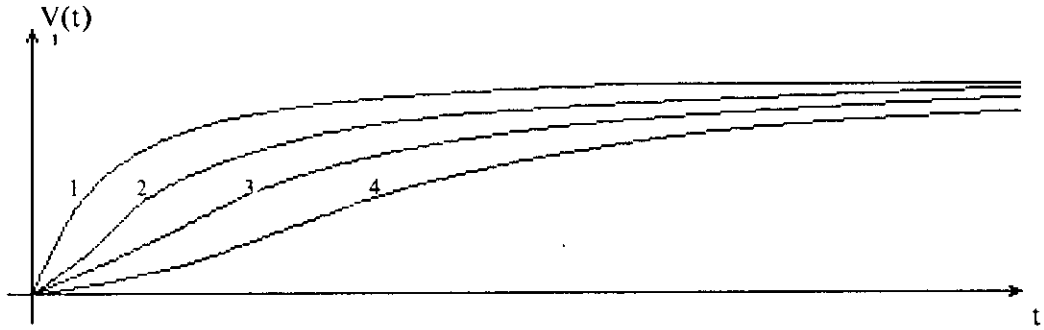


Fig. 11c. Response to Step Function Input

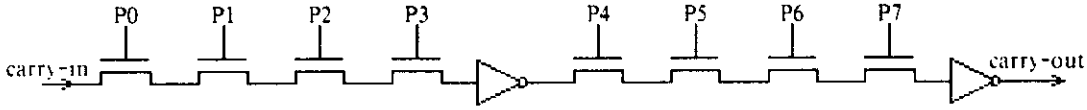


Fig. 11d. Minimizing Delay by Interposing Inverters

Pullup/Pulldown Ratios for Inverting Logic Coupled by Pass Transistors

Early in this chapter we found that when an inverting logic stage directly drives another such stage, a pullup to pulldown ratio $Z_{pu}/Z_{pd} = (L_{pu}/W_{pu})/(L_{pd}/W_{pd})$ of 4:1 yields equal inverter margins and also provides an output sufficiently less than V_{th} for an input equal to VDD. Rather than coupling inverting logic stages directly, we often couple them with pass transistors for the reasons developed in the preceding section, thus affecting the required pullup to pulldown ratio.

Figure 12a shows two inverters connected by a pass transistor. If the output of the first inverter nears VDD, the input of the second inverter can rise at most to $(VDD - V_{thp})$, where V_{thp} is the threshold of the pass transistor. Even with VDD on its gate, the pass transistor opens the connecting path once its input side rises above $(VDD - V_{thp})$. For the second inverter to have its output driven as low with an input of $(VDD - V_{thp})$ as would a standard inverter with an input of VDD, it must have a larger pullup to pulldown ratio, calculated as follows.

With inputs near VDD, the pullups of inverters are in saturation, and the pulldowns are in the resistive region. Figure 12b shows equivalent circuits for two inverters, one with VDD as input, the other with $(VDD - V_{thp})$. For their output voltages to be equal under these conditions, I_1R_1 must equal I_2R_2, and substituting from eq.3a and eq.5 we find:

$$(Z_{pu1}/Z_{pd1})(VDD - V_{th}) = (Z_{pu2}/Z_{pd2})(VDD - V_{th} - V_{thp})$$

Since V_{th} of the pulldowns ~ 0.2VDD, and V_{thp} of the pass transistor ~ 0.3VDD due to the body effect, then $Z_{pu2}/Z_{pd2} \sim 2Z_{pu1}/Z_{pd1}$. Thus a ratio $(L_{pu}/W_{pu})/(L_{pd}/W_{pd}) = 8$ is usually used for inverting logic stages placed as level restorers between sections of pass transistor logic.

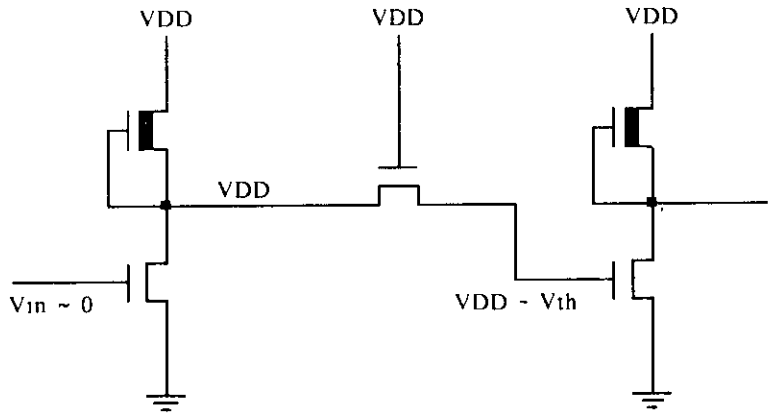


Fig. 12a. Inverters Coupled by Pass Transistor

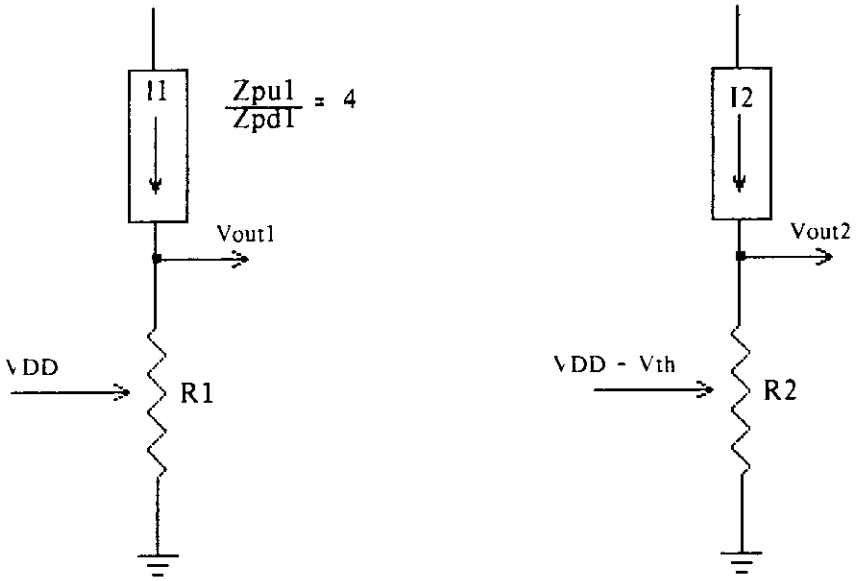


Fig. 12b. For Vout2 = Vout1, Zpu2/Zpd2 = 8

Transit Times and Clock Periods

In chapter 3 we will develop a system design methodology in which we will be able to construct and estimate the performance of arbitrarily complex digital systems, using only the basic circuit forms presented in the preceding sections. The basic *system* building block in the design methodology is a register to register transfer through combinational logic, implemented with pass transistors and inverting logic stages. Using the basic ideas already presented, we may anticipate the results of that chapter in order to estimate the maximum clocking frequency of such systems.

The design methodology uses a two-phase non-overlapping clock scheme, described in detail in chapter 3. During the first clock phase, data passes from one register, through combinational logic stages and pass transistors to a second register. During the second clock phase, data passes from the second register through still more logic and pass transistors to a third (or possibly back to the first) register. The data storage registers are implemented by using charge stored on the input gates of inverting logic stages, the charge being isolated by pass transistors controlled by clock signals, as described in full detail in chapter 3.

Since pass transistors are used to connect inverting logic stages, inverter ratios of $k \sim 8$ are required. If the combinational logic between registers is implemented using only pass transistors, and if the delays through the pass transistors have been carefully matched to those of the inverting logic stages, the total delay will be twice that of the simple $k = 8$ inverter. In the absence of strays, the $k = 8$ inverters have a maximum delay (in the case of the output rising towards VDD) of 8τ, and hence a minimum of 16τ must be allowed for the inverter plus logic delay. However, in most designs the stray capacitance is at least equal to that inherent in the circuit. Thus the minimum time required for one such operation is $\sim 30\tau$. Control lines to the combinational logic and pass transistors each typically drive the gates of 10 to 30 transistors. Even when using a super buffer driver, the delay introduced by this fan out is at least the minimum driving time for a capacitive load. With $Y = 30$, this time is $\sim 9\tau$. To this we must add an 8τ inverter delay for operation of the drivers.

Thus the total time for one clock phase is $\sim 50\tau$. Since two clock phases are required per cycle, a minimum clocking period of $\sim 100\tau$ is required for system designed in this way. In 1978, $\tau \sim 0.3$ nanoseconds, and clocking periods of 30 to 50 ns are achievable in carefully structured integrated systems where successive stages are in close physical proximity. If it is necessary to communicate data over long distances, longer periods are required.

Properties of Cross-Coupled Circuits

In many control sequencing and data storage applications, memory cells and registers are built using two inverters driving each other, as shown in figure 13a. This circuit can be set in either the state where V_1 is high and V_2 is low, or in the state where V_1 is low and V_2 is high. In either case, the condition is stable and will not change to the other condition unless it is forced there through some external means. The detailed methods of setting such cross-coupled circuits into one state or another will be discussed in detail later. However, it is important at the present time to understand the time evolution of signals impressed upon cross-coupled circuits, since they exhibit properties different from circuits not having a feedback path from their output to an input.

We have seen that there exists a voltage at which the output of an inverter is approximately equal to its input voltage. If a cross-coupled circuit is inadvertently placed in a situation where its input voltage is equal to this value, then an unstable equilibrium condition is created where voltages V_1 and V_2 are equal. Since the net current flowing onto either gate is now zero, there is no forcing function driving the system to any voltage other than this equilibrium one, and the circuit can stay in this condition for an indefinite period. However, if either voltage changes, even very slightly, the circuit will leave this unstable equilibrium. For example, if the voltage V_1 is increased from its unstable equilibrium value V_{inv} by a slight amount, this will in time cause a lowering of voltage V_2 , as net current flows from gate 1. This lowering of V_2 will at some later time cause V_1 to increase further. As time goes on, the circuit will feedback on itself until it rests in a stable equilibrium state.

The possibility of such unstable equilibria in cross coupled circuits has important system implications[2], as we will later see. For this reason, we will make a fairly detailed analysis of this circuit's behavior near the metastable state. While it is not essential that the reader follow all the details of the analysis, the final result should be studied carefully. The time constant of the final result depends in detail on the regions of operation of the transistors near the metastable state, as given in the following analysis. However, the exponential form of the result follows simply from the fact that the forcing function pushing the voltage away from the metastable point is proportional to the voltage's distance away from that point. This general behavior is characteristic of bistable storage elements in any technology. However, more complex waveforms are observed in logic families having more than one time constant per stage.

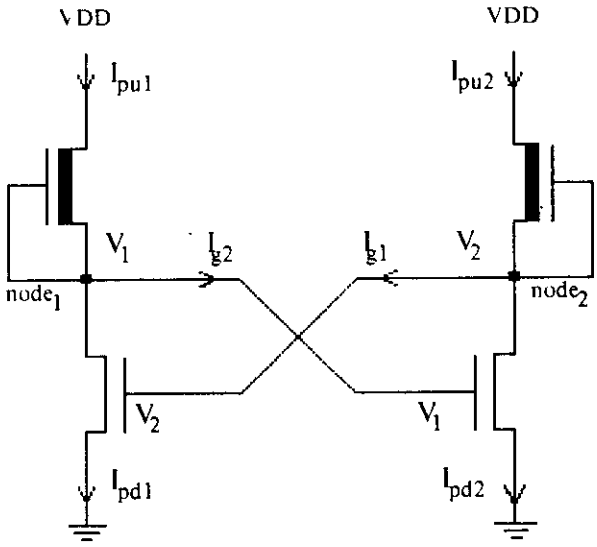


Fig. 13a. Cross Coupled Inverters

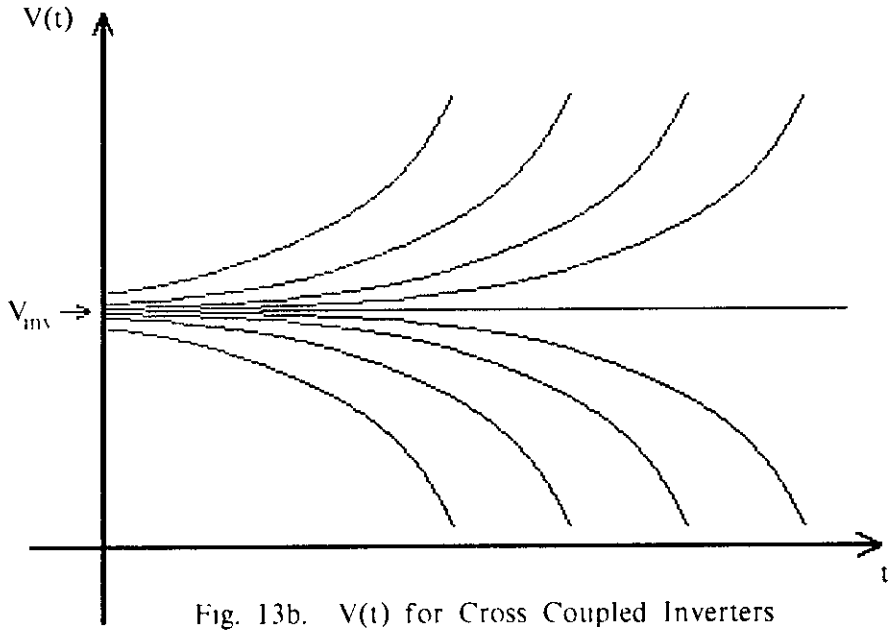


Fig. 13b. V(t) for Cross Coupled Inverters

The time evolution of this process can be traced as follows. At the unstable equilibrium, the current in the pullups equals that in the pulldowns, and is some constant, k_1, times $(V_{inv}-V_{th})^2$. If V_1 is then changed by some small ΔV_1 to V_{init}, I_{pu2} remains constant but I_{pd2} changes immediately, producing a non-zero I_{g1}:

$$I_{g1} = I_{pu2} - I_{pd2} = k_1[(V_{inv}-V_{th})^2 - (V_{inv}+\Delta V_1-V_{th}^2)]$$

For small ΔV_1, $I_{g1} = -2k_1(V_{inv}-V_{th})\Delta V_1$. More precisely, since I_{g1} = function(V_1,V_2), then near V_{inv}:

$$\partial I_{g1}/\partial V_1 = -2k_1(V_{inv}-V_{th})$$

Noting that the pullups are not quite in saturation, but are in the resistive region, and:

$$\partial I_{g1}/\partial V_2 = -1/R_{pu} ,$$

where R_{pu} = effective resistance of the pullup near V_{inv}. Noting that $I_{g1} = C_g dV_{g2}/dt$, we find that:

$$dI_{g1}/dt = -2k_1(V_{inv}-V_{th})[dV_1/dt] - (1/R_{pu})[dV_2/dt] = C_g[d^2V_2/dt^2]$$

Evaluating the constants in this equation yields $-k_1(V_{inv}-V_{th}) = C_g/\tau_o$, where τ_o is the saturation transit time of the pulldowns for t near zero. If we assume a pullup/pulldown Z ratio of 4:1, and consider the operating conditions near t = 0, then evaluating the effective resistance of the pullups in terms of the parameters of the pulldowns yields $1/R_{pu} \sim C_g/\tau_o$. Therefore:

$$-(2/\tau_o)[dV_1/dt] - (1/\tau_o)[dV_2/dt] = d^2V_2/dt^2$$

Similarly:
$$-(2/\tau_o)[dV_2/dt] - (1/\tau_o)[dV_1/dt] = d^2V_1/dt^2$$

Near t = 0, dV_1/dt approximately equals $-dV_2/dt$, and therefore:

$$d^2V_1/dt^2 = -(1/\tau_o)\, dV_2/dt = (1/\tau_o)^2\, V_1 + \text{const.} \qquad \text{[eq.16a]}$$

The solution to eq. 16a is an exponential diverging from the equilibrium voltage V_{inv}, with a time constant $\tau_o/2$ equal to 1/2 the pulldown delay time. Note that the solution given in eq.16b satisfies the conditions that $V(0) = V_{init}$, and $V(t)$ = constant if $V_{init} = V_{inv}$:

$$V_1(t) = V_{inv} + (V_{init} - V_{inv})\, e^{t/\tau_o} \qquad \text{[eq.16b]}$$

The above analysis applies to cross coupled circuits in the absence of noise. Noise unavoidably present in the circuit spreads the input voltage into a band from which such an unstable equilibrium can statistically be initiated. The width of this band is equal to the noise amplitude. Any timing condition which causes the input voltage to settle in this band has some probability of causing a balanced condition, from which the circuit may require an arbitrarly long time to recover. The time evolution of such a system is shown in Fig. 13b, for several initial voltages near V_{inv}. The time for the cross-coupled system to reach one of its equilibria is thus logarithmic in the displacement from V_{inv}, and is given approximately by eq. 16c:

$$t \sim \tau_o \ln[V_{inv}/(V_{init}-V_{inv})] \qquad \text{[eq.16c]}$$

Effects of Scaling Down the Dimensions of MOS Circuits and Systems

This section examines the effects on important system parameters resulting from scaling down all dimensions of an integrated system, including those vertical to the surface, by dividing them by a constant factor α. The voltage is likewise scaled down by dividing by the same constant factor α. Using this convention, all electric fields in the circuit will remain constant and hence many non-linear factors affecting performance will not change as they would if a more complex scaling were used.

Figure 14a. shows a MOSFET of dimensions L, W, D, with a (Vgs - Vth) = V. Figure 14b. shows a MOSFET similar to that in figure 14a., but of dimensions $L' = L/\alpha$, $W' = W/\alpha$, $D' = D/\alpha$, and $V' = V/\alpha$. Refer to equations 1., 2., and 3. From these equations we will find that as the scale down factor α is increased, the transit time, the gate capacitance, and drain to source current of every individual transistor in the system scale down proportionally, as follows:

$$\tau \propto L^2/V, \quad \tau'/\tau = [(L/\alpha)^2/(V/\alpha)]/[L^2/V], \quad \therefore \quad \tau' = \tau/\alpha$$

$$C \propto LW/D, \quad C'/C = (L/\alpha)(W/\alpha)/(D/\alpha)]/[LW/D], \quad \therefore \quad C' = C/\alpha$$

$$I \propto WV^2/LD, \quad I'/I = [(WV^2/\alpha^3)/(LD/\alpha^2)]/[WV^2/LD], \quad \therefore \quad I' = I/\alpha$$

Switching power, P_{sw}, is the energy stored on the capacitance of a given device divided by the *clock period*, or time between successive charging and discharging of the capacitance. A system's clock period is proportional to the τ of its smallest devices. As devices are made smaller and faster, the clock period is proportionally shortened. Also, the dc power, P_{dc}, dissipated by any static circuit equals I times V. Therefore,

$$P_{sw} \propto CV^2/\tau \propto WV^3/DL, \qquad \therefore \quad P_{sw}' = P_{sw}/\alpha^2$$

$$P_{dc} = IV, \qquad \therefore \quad P_{dc}' = P_{dc}/\alpha^2$$

Both the switching power and static power per device scale down as $1/\alpha^2$. The average dc power for most systems can be approximated by adding the total P_{sw} to one-half the total dc power which would result if all level restoring logic pulldowns were turned on. The contribution of pass transistor logic to the average dc power drawn by the system is due to the switching power consumed by the driving circuits which charge and discharge the pass transistor control gates.

The switching energy per device, E_{sw}, is an important metric of device performance. It is equal to the power consumed by the device at maximum clock frequency multiplied by the device delay, and scales down as follows:

$$E_{sw} \propto CV^2, \qquad \therefore \quad E_{sw}' = E_{sw}/\alpha^3$$

The following table summarizes values of the important system parameters for current technology, and for a future technology near the limits imposed by physical law:

	1978	**19XX**
Minimum Feature Size:	6 μm	0.3 μm
τ:	~0.3 ns	~0.02 ns
Esw:	$\sim 10^{-12}$J	$\sim 2 \times 10^{-16}$J
System Clock Period: [see earlier section]	~30 to 50 ns	~2 to 4 ns

A more detailed plot of the channel conductance of an MOS transistor near the threshold voltage is shown in figure 15. Below the nominal threshold, the conductance (1/R) is not in reality zero, but depends on gate voltage and temperature as follows:

$$1/R \propto e^{(V_{gs}-V_{th})/(kT/q)},$$

where T is the absolute temperature, q is the charge on the electron, and k is Boltzmann's constant. At room temperature, $kT/q \sim 0.025$ volts. At present threshold voltages, as in the right curve in figure 15., an off device is below threshold by perhaps 20 kT/q, i.e. by about 0.5 volts, and its conductance is decreased by a factor of the order of ten million. Said another way, if the device is used as a pass transistor, a quantity of charge which takes a time τ to pass through the on device, will take a time on the order of $10^7\tau$ to "leak" through the off device. The use of pass transistors switches to isolate and "dynamically store" charge on circuit nodes is common in many memory applications using 1978 transistor dimensions. However, if the threshold voltage is scaled down by a factor of perhaps 5, as shown in the left plot in figure 15., then an off transistor is only 4kT/q below threshold. Therefore its conductance when "off" is only a factor of 100 or so less than when it is "on". Charge stored dynamicaly on a circuit node by the transistor when "on", will safely remain on that node for only a few typical system clock periods. The charge will not remain on the node for a very large number of periods as in present memory devices using this technique.

Now, suppose we scale down an entire integrated system by a scale down factor of $\alpha = 10$. The resulting system will have one hundred times the number of circuits per unit area. The total power per unit area remains constant. All voltages are reduced by the factor of 10. The current per unit area is increased by a factor of 10. The time delay per stage is decreased by a factor of 10. The power-delay product decreases by a factor of 1000.

This is a rather attractive scaling in all ways except for the current density. The delivery of the required average dc current presents an important obstacle to scaling. This current must be carried to the various circuits in the system on metal conductors, in order that the voltage drop from the off-chip source to the on-chip subsystems will not be excessive. Metal paths have an upper current density limit imposed by a phenomenon called metal migration, discussed further in chapter 2. Many metal paths in today's integrated circuits are already operated near their current density limit. As the above type of scaling is applied to a system, the conductors get narrower, but still deliver the same current on the average to the circuits supplied by them.

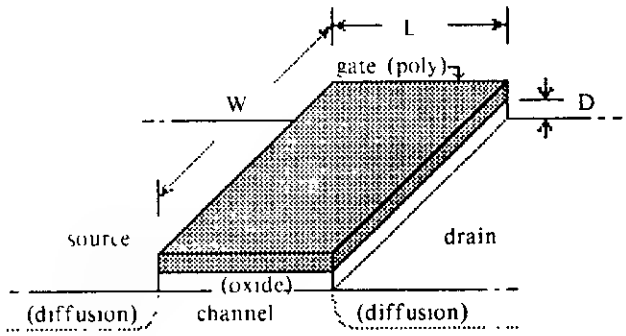


Fig. 14. MOSFET, 1978

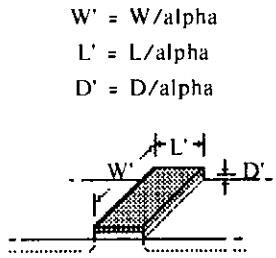


Fig. 14b. MOSFET Scaled Down by Alpha, 19XX

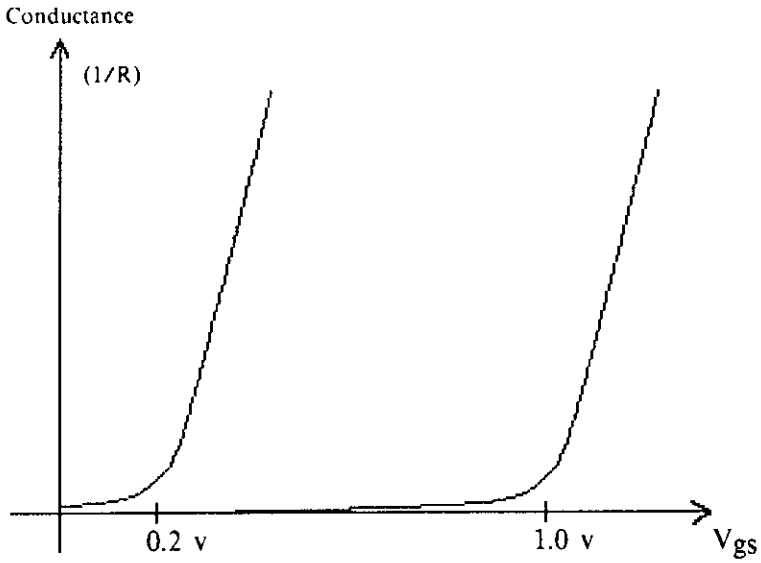


Fig. 15. Conductance as a Function of Threshold Voltage

Therefore, it will be necessary to find ways of decreasing system current requirements to approximately a constant current per unit area relative to the present current densities. In n-channel silicon gate technology, this objective can be partially achieved by using pass transistor logic in as many places as possible and avoiding restoring logic except where it is absolutely necessary. Numerous examples of this sort of design are given later in this text. This design approach also has the advantages of tending to minimize delay per unit function and to maximize logic functions per unit area. However, when scaled down to submicron size, the pass transistors will suffer from the subthreshold current problem. It is possible that when the fabrication technologies have been developed to enable scaling down to submicron devices, a technology such as complementary MOS, which does not draw any dc current, may be preferable to the nMOS technology used to illustrate this monograph. However, even if this occurs, the methodology developed in the text can still be applied in the design of integrated systems in that technology.

The limit to the kind of scaling described above occurs when the devices created are no longer able to perform the switching function. To perform the switching function, the ratio of transistor on to off conductance must be >>1, and therefore the voltage operating the circuit must be many times kT/q. For this reason, even circuits optimized for operation at the lowest possible supply voltages still require a VDD of ~ 0.5 volts. Devices in 1978 operate with a VDD of approximately five volts and minimum channel lengths of approximately six microns. Therefore, the kind of scaling we have envisioned here will take us to devices with approximately one half micron channel lengths and current densities approximately ten times what they are today. Power per unit area will remain constant over that range. Smaller devices might be built but must be used without lowering the voltage any further. Consequently the power per unit area will increase. Finally, there appears to be a fundamental limit[3] of approximately one quarter micron channel length, where certain physical effects such as the tunneling through the gate oxide, and fluctuations in the positions of impurities in the depletion layers, begin to make the devices of smaller dimension unworkable.

References

1. W. M. Penney, L. Lau, Eds., "MOS Integrated Circuits", Van Nostrand Reinhold, 1972, pp.60-85.

2. T. J. Chaney, C .E. Molnar, "Anomalous Behavior of Synchronizer and Arbiter Circuits", *IEEE Transactions of Computers*, April 1973, pp. 421-422.

3. B. Hoeneisen, C. A. Mead, "Fundamental Limitations in Micro-electronics--I. MOS Technology", *Solid-State Electronics*, Vol.15, 1972, pp. 819-829.

4a. T. Kilburn, D. B. G. Edwards, D. Aspinall, "A Parallel Arithmetic Unit Using a Saturated Transistor Fast-Carry Circuit", Proc. IEE, Pt. B, vol. 107, pp.573-584, November, 1960.

4b. Staff of the Computation Lab, "Description of a Relay Calculator", Annals of the Harvard Computation Lab, vol. 24, Harvard University Press, 1949.

5. T. K. Young, R. W. Dutton, "MINI-MSINC - A Minicomputer Simulator for MOS Circuits with Modular Built-in Model", Stanford Electronics Laboratories, Technical Report No. 5013-1, March 1976.

6. L. Lagel, D. Pederson,"Simulation Program with Integrated Circuit Emphasis (SPICE)", 16th Midwest Symposium on Circuit Theory, Waterloo, Ontario, Apr. 12, 1973.

Reading References

R1. J. F. Gibbons, "Semiconductor Electronics", McGraw-Hill, 1966, is a classic text containing a good introduction to basic semiconductor theory.

R2. W.M. Penney, L. Lau, Eds., "MOS Integrated Circuits", Van Nostrand Reinhold,1972, is a good, general text on MOS devices and circuits.

R3. R. W. Keyes, "Physical Limits in Digital Electronics", *Proceedings of the IEEE*, Vol. 63, No. 5, May 1975, pp. 740-767, is an excellent invited survey paper on this topic.

R4. P. Richman, "MOS Field-Effect Transistors and Integrated Circuits", Wiley, 1973, provides an excellent discussion of the physics and fabrication of MOS devices.

Chapter 2: Integrated System Fabrication

Sections:

The series of steps by which a geometric pattern or set of geometric patterns is transformed into an operating integrated system is called a *wafer fabrication process*, or simply a *process*. An integrated system in MOS technology consists of a number of superimposed layers of conducting, insulating, and transistor forming materials. By arranging predetermined geometric shapes in each of these layers, a system of the required function may be constructed. The task of the integrated system designer is to devise the geometric shapes and their locations in each of the various layers of the system. The task of the process itself is to create the layers and transfer into each of them the geometric shapes determined by the system design.

Modern wafer fabrication is probably the most exacting production process ever developed. Since the 1950's, enormous human resources have been expended by the industry to perfect the myriad of details involved. The impurities in materials and chemical reagents are measured in parts per billion. Dimensions are controlled to a few parts per million. Each step has been carefully devised to produce some circuit feature with the minimum possible deviation from the ideal behavior. The results have been little short of spectacular: chips with many tens of thousands of transistors be produced for under ten dollars. In addition, wafer fabrication has reached a level of maturity where the system designer need not be concerned with the fine details of its execution. The following sections present a broad overview sufficient to convey the ideas involved, and in particular those relevant for system design. Our formulation of the basic concepts anticipates the evolution of the technology towards ever finer dimensions.

In this chapter we describe the patterning sequence and how it is applied in a simple, specific integrated system process: nMOS. A number of other topics are covered which are related to the processing technology, or are closely tied to the properties of the underlying materials.

Patterning

The overall fabrication process consists of the successive *patterning* of a particular *sequence of layers*. The *patterning* steps by which geometrical shapes are transferred into a layer of the final system, is very similar for each of the layers. The overall process is more easily visualized if we first describe the details of patterning one layer. We can then describe the particular sequence of layers used in the process to build up an integrated system, without repeating the details of patterning for each of the layers.

A common step in many processes is the creation of a silicon dioxide insulating layer on the surface of a silicon wafer, and the selective removal of sections of the insulating layer exposing the underlying silicon. We will use this step for our patterning example. The step begins with a bare polished silicon wafer, shown in cross section in figure 1. The wafer is exposed to oxygen in a high temperature furnace to grow a uniform layer of silicon dioxide on its surface, as shown in figure 2. After the wafer is cooled, it is coated with a thin film of organic resist material as shown in figure 3. The resist is thoroughly dried and baked to insure its integrity. The wafer is now ready to begin the patterning.

At the time of wafer fabrication the pattern to be transferred to the wafer surface exists as a *mask*. A mask is merely a transparent support material coated with a thin layer of opaque material. Certain portions of the opaque material are removed, leaving opaque material on the mask in the precise pattern required on the silicon surface. Such a mask with the desired pattern engraved upon it is brought face down into close proximity with the wafer surface as shown in figure 4. The dark areas of opaque material on the surface of the mask are located where it is desired to leave silicon dioxide on the surface of the silicon. Openings in the mask correspond to areas where it is desired to remove silicon dioxide from the silicon surface. When the mask has been brought firmly into proximity with the wafer itself, its back surface is flooded with an intense source of ionizing radiation such as ultraviolet light or low energy x-rays. The radiation is stopped in areas where the mask has opaque material on its surface. Where there is no opaque material on the mask surface, the ionizing radiation radiation passes on through and into the resist, the silicon dioxide, and silicon. While the ionizing radiation has little effect on the silicon dioxide and the silicon, it breaks down the molecular structure of the resist* into considerably smaller molecules.

* We have chosen to illustrate this text using *positive* resist, i.e. the resist material remaining after exposure and development corresopponds to the opaque mask areas. Negative resists are also in common use. Positive resists are typically workable to finer feature sizes, and are likely to become dominant as the technology progresses.

Figure 1. Bare Wafer

Figure 2. Oxidation

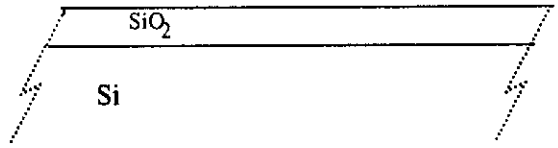


Figure 3. Coat w/Resist

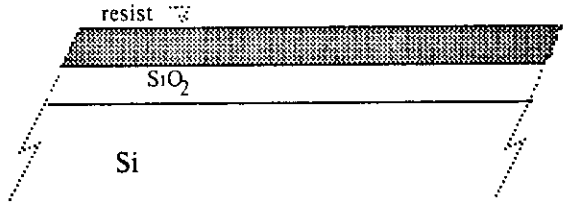
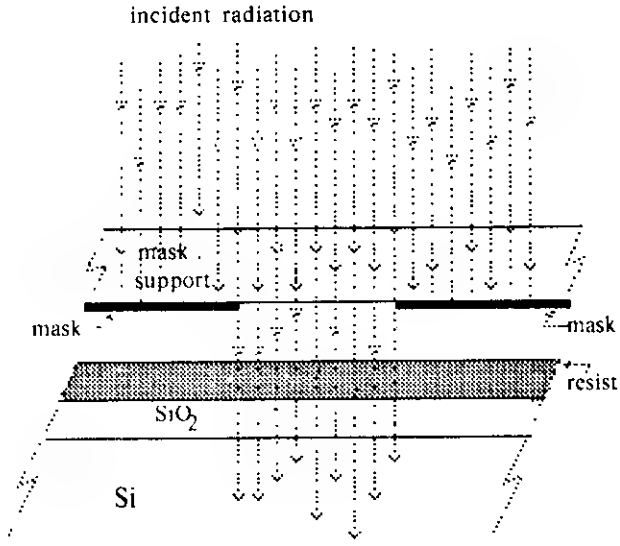


Figure 4. Mask & Expose



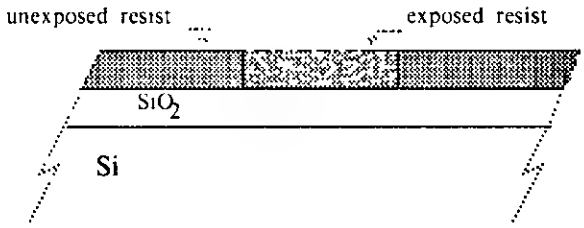


Figure 5. Exposed Resist

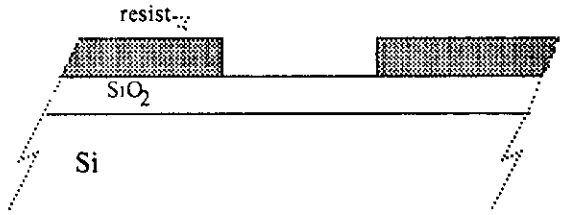


Figure 6. Develop Resist

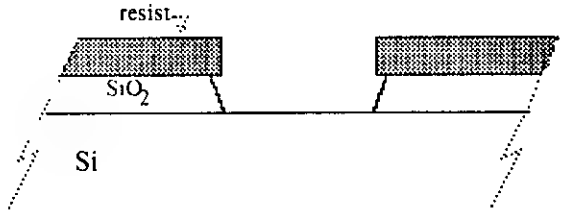


Figure 7. Etch

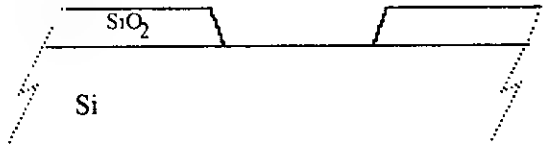


Figure 8. Remove Resist

After exposure to the ionizing radiation, the wafer has the characteristics shown in figure 5. In areas exposed to the radiation, the resist molecules have been broken down to much lighter molecular weight than that of unexposed resist molecules. The solubility of organic molecules in various organic solvents is a very steep function of the molecular weight of the molecules. Hence, it is possible to dissolve exposed resist material in solvents which will not dissolve the unexposed resist material. In this way the resist can be "developed" as shown in figure 6 by merely immersing the silicon wafer in a suitable solvent.

Thus far, the pattern originally existing as a set of opaque geometries on the mask surface has been transferred into a corresponding pattern in the organic resist material on the surface of the silicon dioxide. This same pattern can now be transferred to the silicon dioxide itself by exposing the wafer to a material which will etch silicon dioxide but not attack either the organic resist material or the silicon wafer surface. This etching step is usually done with hydrofluoric acid, which easily dissolves silicon dioxide. However, organic materials are very resistant to hydrofluoric acid, and it is incapable of etching the surface of silicon. The result of this etching step is shown in figure 7.

The final step in patterning is removal of the remaining organic resist material. Three techniques have been used to remove resist materials. Strong organic solvents will dissolve even unexposed resist material. Strong acids such as chromic acid actively attack organics. The wafer can be exposed to atomic oxygen which will oxidize away any organic materials present on its surface. Once the resist material is removed, the finished pattern on the wafer surface is as shown in figure 8. Notice that we have transferred the geometric pattern which originally existed on the surface of the mask directly into the silicon dioxide on the wafer surface. While a foreign material was present on the wafer surface during the patterning process, it has now disappeared and the only materials present are those which will be part of the finished wafer.

A similar sequence of steps is used to selectively pattern each of the layers of the integrated system. These differ only in the details of the etchants used, etc. Thus as we study the processing of the various layers, the reader need not visualize all the details of the patterning sequence for each layer, but only recognize that a mask pattern for a layer can be transferred into a pattern in the material of that layer.

Scaling of Patterning Technology

As discussed in chapter 1, semiconductor devices could be at least an order of magnitude smaller in linear dimension than those typically manufactured in 1977 and still function correctly. The fundamental dimensional limitation is approximately a one quarter micron channel length, corresponding to a length unit λ (to be discussed under design rules) of approximately 0.1 micron. This limitation appears to apply to both bipolar and MOS technologies. It has been possible for several years to create sub-micron lines using electron beam and x-ray techniques, and there is considerable research and development under way to bring these patterning technologies into general manufacturing use. It appears that there are no fundamental barriers preventing creation of patterns for ultimately small devices. A more detailed discussion of the techniques involved is given in chapter 4.

The Silicon Gate n-Channel MOS Process

We now describe the particular sequence of patterned layers used to build up nMOS integrated circuits and systems. Figures 9 through 14 illustrate a simple but complete sequence of patterning and processing steps which are sufficient to fabricate a complete, integrated system. The example follows the fabrication of one simple circuit within a system, but all other circuits are simultaneously implemented by the same process. The example used is the basic inverter circuit. The top illustration in figures 9 through 14 shows the top view of the layers of the circuit layout. The lower illustration in each of those figures shows the cross section through the cut indicated by the downward arrows. The vertical scale in these cross sections has been greatly exaggerated for illustrative purposes.

The opening in the opaque material of the first mask is shown by the green outline in the top portion of figure 9. This opening exposes all areas that will be eventually be the diffusion level. It includes the sources and drains of all transistors in the circuit, together with the transistor gate areas, and any diffusion level circuit interconnection paths. This mask is used for the first step in the process, the patterning of silicon dioxide on silicon as described in the previous section. The resulting cross section is shown in the lower portion of figure 9.

Fig.9. Patterning SiO_2

Fig.10. Patterning Ion Implantation

Fig.11. Patterning Polysilicon

Fig.12. Placing Diffused Region

Fig.13. Placing Contact Cuts

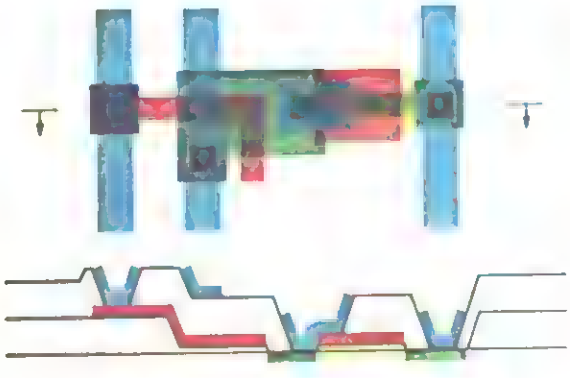


Fig.14. Patterning the Metal Layer

The second step in the process is to differentiate transistors which are normally "on" (depletion mode) from those which are normally "off" (enhancement mode). This is done by overcoating the wafer with resist material, exposing the resist material through openings in a second mask, and developing it in the manner shown in figure 10. This patterning step leaves an opening in the resist material over the area to be selectively turned into depletion mode transistors. The actual conversion of the underlying silicon is then done by implanting ions of arsenic or antimony into the silicon surface. The resist material, where present, acts to prevent the ions from reaching the silicon surface. Therefore, ions are only implanted in the silicon area free of resist. The implanted layer, which causes a slight n-type conductivity in the underlying silicon, is shown by the yellow box in figure 10. Once the depletion areas are defined, the resist material is removed from the surface of the wafer.

The wafer is then heated while exposed to oxygen, to grow a very thin layer of silicon dioxide over its entire surface. It is then entirely coated with a thin layer of polycrystalline silicon, usually called *polysilicon* or *poly* for short. Note that this polysilicon layer is everywhere insulated from the underlying materials by the layer of thin oxide, and additionally by thicker oxide in some areas. The polysilicon will form the gates of all the transistors in the circuit and will also serve as a second layer for circuit interconnections. A third mask is used to pattern the polysilicon by steps similar to those previously described, with the result shown in red in figure 11. The left-most polysilicon area will function as the gate of the pull down transistor of the inverter we are constructing, while the square to the right will function as the gate of the depletion mode pull up transistor.

Once the polysilicon areas have been defined, n-type regions can be diffused into the p-type silicon substrate, forming the sources and drains of the transistors and the first level of interconnections. This step is done by first removing the thin gate oxide in all areas not covered by the (red) polysilicon. The wafer is then exposed to n-type impurities such as arsenic, antimony or phosphorus at high temperature for a sufficient period of time to allow these impurities to convert the exposed underlying silicon to n-type material. The areas of resulting n-type material are shown in green. Notice, in the *cross section* of figure 12., that the red polysilicon area and the thin oxide under it act to prevent impurities from diffusing into the underlying silicon. Therefore the impurities reach the silicon substrate only in areas not covered by the polysilicon and not overlain by the thick original oxide. In this way the active transistor area is formed in all places where the patterned polysilicon overlies the thin oxide area defined in the previous step. The diffusion level sources and drains of the transistors are automatically filled in between the polysilicon areas and extend

up to the edges of the thick oxide areas. The major advantage of the silicon gate process is that it does not require a critical alignment between a mask which defines the green source and drain areas and a separate mask which defines the gate areas. Rather, the transistors are formed by the intersection of the two masks, and the conducting n-type diffused regions are formed in all areas where the green mask is not covered by the red mask.

All the transistors of the basic inverter circuit are now defined. Connections must now be made to the input gate, between the gate and source of the pullup, and to VDD and GND. These interconnections will be made with a metal layer that can make contact to both the diffused areas and the polycrystalline areas. However, in order to ensure that the metal does not make contact to underlying areas except where intended, another layer of insulating oxide is coated over the entire circuit. At the places where the overlying metal is to make contact to either the polysilicon or the diffused areas, the overlying oxide is selectively removed by the patterning process as previously described. The result of coating the wafer with the overlying oxide and removing this oxide in places where contacts are desired, is shown in figure 13. In the top view, the black areas are those defined by openings in the contact mask, the fourth in the process's sequence of mask patterns. In cross section notice that in the contact areas all oxide has been removed down to either the polycrystalline silicon or the diffused area.

Once the overlying oxide has been patterned in this way, the entire wafer is coated with metal, usually aluminum, and the metal is patterned with a fifth mask to form the conducting areas required by the circuit. The top view in figure 14 shows three metal lines running vertically, the left most connecting to the input gate of the inverter, the center one being ground, and the right-most one forming the VDD connection to the inverter. The peculiar structure* formed by the metal square slightly to the right of center connects the polysilicon gate of the depletion mode pull up transistor to its source and to the drain of the pull down transistor. Rather than making two separate contacts from the metal line to the pullup's polysilicon gate region and to the adjacent diffusion region, area can be conserved by coalescing the contacts into the compact arrangement shown. This geometrical arrangement is known as a *butting contact* and will be used extensively throughout the text.

*In general, it is good practice to avoid placing contacts over active transistor area whenever possible. However, butting contacts in the location shown here reduce the area and simplify the geometry of the basic inverter and many other circuits, and have been so placed by the authors in many systems run on a number of commercial fabrication processes without ill effects. A more conservative approach would be to place the butting contact adjacent to, rather than over, the active pullup area.

The inherent properties of the silicon gate process allow the blue metal layer to cross over either the red polysilicon layer or the green diffused areas, without making contact unless one is specifically provided. The red polysilicon areas, however, cannot cross the green diffused areas without forming a transistor. The transistors formed by the intersection of these two masks can be either enhancement mode if no yellow implantation is provided, or depletion mode if such an implantation is provided. Hence, the enhancement mode transistors are defined by the intersection of the green and red masks while the depletion mode transistors are defined by the intersection of the green, red and yellow masks.

If we wish to fabricate only a small number of prototype system chips and to have access to the metal level for the probing of test points, the wafer fabrication sequence can be terminated at this step. However, when fabricating large numbers of chips of a debugged design, the wafer surface is usually coated with another layer of oxide. This step, called *overglassing*, provides physical protection for the devices in the system. A sixth mask is then used to pattern contact cuts in the overglassing at the locations of relatively large metal contact pads.

Each wafer contains many individual chips. The chips are separated by scribing the wafer surface with a diamond scribe, and then fracturing the wafer along the scribe lines. Each individual chip is then cemented in place in a package, and fine metal wire leads are bonded to the metal contact pads on the chip and to pads in the package which connect with its external pins. A cover is then cemented over the recess in the package which contains the silicon chip, and the completed system is ready for testing and use.

Yield Statistics

Of the large number of individual integrated system chips fabricated on a single silicon wafer, only a fraction will be completely functional. Flaws in the masks, dust particles on the wafer surface, defects in the underlying silicon, etc., all cause certain devices to be less than perfect. With present design techniques, any single flaw of sufficient size will kill an entire chip.

The simplest model for the *yield*, or the fraction of the chips fabricated which do not contain fatal flaws, assumes (naively) that the flaws are randomly distributed over the wafer,

and that one or more flaws anywhere on a chip will cause it to be non-operative. If there are N fatal flaws per unit area, and the area of an individual chip is A, the probability that a chip has n flaws is in the simplest case just given by the Poisson distribution, $P_n(NA)$. The probability of a good chip is:

$$P_0(NA) = e^{-NA} \qquad \text{[eq.1]}$$

While this equation does not accurately represent the detailed behavior of real fabrication processes, it is a good approximate model for estimating the yield of alternative designs. The exponential is such a steep function that a very simple rule is possible: chips with areas many times 1/N will simply never be found without flaws. Areas must be kept less than a few times 1/N if one flaw will kill a system. Design forms may be developed in the future which will permit systems to work even in the presence of flaws. If such forms are developed, the entire notion of yield will be completely changed and much larger chips will be possible.

Once a wafer has been fabricated, each chip must be tested to determine if it is functional. Testing of simple combinatorial logic networks is straightforward and may be done completely. Complete testing of complex systems with internal sequencing is not in general possible, and most integrated system chips manufactured, even at 1978 levels of complexity, are not economically testable even for a small fraction of their possible internal states.

As time passes and the number of devices per chip continues to increase, it becomes important to consider including special functions in the design of integrated systems to improve their testability. The basic problem is to linearize an otherwise combinatorial problem. One approach to this is:

(i) Define the entire system as a set of register to register transfer blocks, i.e. successive stages of storage registers with combinational logic between them.
(ii) Provide for reading and writing from the external world to/from each of the storage registers.

The storage locations are first tested independently for their ability to store data or control information. If all storage locations pass this test, each combinational logic block can be tested separately, by use of its input and output storage locations. Such a test becomes essentially linear in the number of components, and may be accomplished in an acceptable

time period, even for extremely complex systems. However, without access to the individual storage locations, testing rapidly becomes hopeless. For this reason even present day microprocessors are very incompletely tested. When one is used for a while, an apparently new and sudden malfunction may simply be the first occurrence of a particular state of control and data in the system, and thus may represent the first time the device had been "tested" under those conditions.

From experience gained in testing memory parts, it is known that the behavior of one circuit can be influenced by the state of a nearby circuit. For example, a memory cell may be able to remember both a logic-1 and a logic-0 if its neighbor is at a logic-0, but may be able to retain only a logic-0 if its neighbor is at a logic-1. Failures of this type are dependent upon the data patterns present in the system, and are known as *pattern sensitive* failures. In a reasonable (or even an unreasonable) time, it is not possible to exercise even a minute fraction of all the combinations of bit patterns of many integrated systems. What is done instead is to apply our knowledge of the physics of such failures, and construct a *model* for possible failure modes. In the memory example, we may conclude that any flaw not visible optically will be unable to reach beyond the immediate locality of the cell involved. Hence, pattern sensitivity in the behavior of a particular cell may be introduced by other cells in the same row or column of an array of memory cells, or by diagonal nearest neighbors. A test for pattern sensitivity under this model is quite fast, being only slightly worse than a linear function of the number of devices on the chip.

In order to test for pattern sensitive failures, we must construct a physical model for the possible failure mechanisms. This model will inevitably include the physical proximity of other signals. For this reason, any practical test for pattern sensitive failures must be based on a knowledge of the physical location of the various elements of the subsystem being tested. The task of preparing such tests is thus greatly eased by regularity in the design and physical layout of a system.

Scaling of the Processing Technology

In order to have a complete process for sub-micron transistors, it is necessary not only to make patterns in the resist material but to transfer these patterns to the underlying layers in the silicon and silicon dioxide. Traditionally, wet etching processes have been used. However, wet etching processes do not scale well into the sub-micron range. Alternatives are currently being developed which appear workable. Etching with plasmas (i.e. glow discharges of gaseous materials resulting in free ions of great chemical activity) is already used in a number of advanced processing laboratories. It is known that very well controlled etching can be achieved in this way and it seems likely that essentially no wet processing will be used in the construction of sub-micron devices. Ion implantation, an ideal method for achieving controlled doses of impurity ions in the silicon surface, is already a common production technique in essentially all MOS processing facilities. Metal layers for sub-micron processes must be thicker in relationship to their width than today's commercial processing technology allows. A possible solution to this problem may be the use of a process known as ion milling for metal patterning. In this process ions of modest energy sputter away any metal not covered with resist material, yielding much steeper sides on the metal thus patterned than do current wet etching processes.

It appears that the basic technological pieces exist to enable development of a complete patterning and wafer fabrication process at sub-micron dimensions. In reality, the ultimate submicron process will not emerge full-blown, but dimensions will gradually be reduced, as one after another of the myriad of technological difficulties are surmounted. The sketch we have given is rather an artist's conception of the possibility of such an ultimate process. We do believe, however, that the evolution of this process is of fundamental importance to the entire electronics industry.

Design Rules

Perhaps the most powerful attribute of modern wafer fabrication processes is that they are *pattern independent.* That is, there is a clean separation between the processing done during wafer fabrication and the design effort which creates the patterns to be implemented. This separation requires a precise definition to the designer of the capabilities of the processing line. This specification usually takes the form of a set of permissible geometries which may be used by the designer with the knowledge that they are within the resolution of the process itself and that they do not violate the device physics required for proper operation of transistors and interconnections formed by the process. When reduced to their simplest form, such geometrical constraints are called *design rules.* The constraints are of the form of minimum allowable values for certain *widths*, *separations*, *extensions*, and *overlaps* of geometrical objects patterned in the various levels of a system.

As processes have improved over the years, the absolute values of the permissible sizes and spacings of various layers have become progressively smaller. There is no evidence that this trend is abating. In fact, there is every reason to believe that at least another order of magnitude of shrinkage in linear dimensions is possible. For this reason we present a set of design rules in dimensionless form, as constraints on the allowable ratios of certain distances to a basic length unit. The basic unit of length measurement used is equal to the fundamental resolution of the process itself. This is the distance by which a geometrical feature on any one layer may stray from another geometrical feature on the same or on another layer, all processing factors considered and an appropriate safety factor added. It is set by phenomena such as overetching, misalignment between mask levels, distortion of the silicon wafer ("runout") due to high temperature processing, over or underexposure of resist, etc. All dimensions are given in terms of this elementary distance unit, which we refer to as the *length-unit*, λ. In 1978 the length-unit $\lambda \sim 3$ microns. One micron (μm) = 10^{-6} meters.

The rules given below have been abstracted from a number of processes over a range of values of λ, corresponding to different points in time at different fabrication areas. They represent somewhat of a "least common denominator " likely to be representative of nMOS design rules for a reasonable period of time, as the value of λ decreases in the future.

A typical minimum for the line width W_d of the diffused regions is 2λ, as shown in figure 15. The spacing required between two electrically separate diffused regions is a parameter which depends not merely upon the geometric resolution of the process, but also upon the physics of the devices formed. If two diffused regions pass too close together, the depletion layers associated with the junctions formed by these regions may overlap and result in a current flowing between the two regions when none was intended. In typical processes a safe rule of thumb is to allow 3λ of separation, S_{dd}, between any two diffused regions which are unconnected, as shown in figure 16. The width of a depletion layer associated with any diffused region depends upon the voltage on the region. If one of the regions is at ground potential, its depletion layer will of necessity be quite thin. In addition some processes provide a heavier doping level at the surface of the wafer between the diffused areas in order to alleviate the problem of overlap of depletion layers. In cases where either very low voltage exists on both diffused regions or where a heavily doped region has been implanted in the surface between the diffused areas, it is often possible to space diffused areas 2λ apart. However, this should not be done without carefully checking the actual process by which the design is to be fabricated.

The minimum for the width W_p of polysilicon lines is similarly 2λ. No depletion layers are associated with polysilicon lines, and therefore the separation, S_{pp}, of two such lines may be as litle as 2λ. These rules are illustrated in figures 17 and 18.

We have so far considered the diffused and polysilicon layers separately. Another type of design rule concerns how the two layers interact with each other. Figure 19 shows a situation where a diffused line is running parallel to an independent polysilicon line, to which it is not anywhere connected. The only consideration here is that the two unconnected lines not overlap. If they did they would form an unwanted capacitor. Avoidance of this overlap requires a separation S_{pd} of only λ between the two regions as shown in figure 19. A slightly more complex situation is shown in figure 20, where a polysilicon gate area intentionally crosses a diffused area, thereby forming a transistor. In order to make absolutely sure that the diffused region does not reach around the end of the gate and short out the drain to source path of the transistor with a thin diffused area, it is necessary for the polysilicon gate to extend a distance E_{pd} of at least 2λ beyond the nominal boundary of the diffused area, as shown in figure 20.

A composite of several of these design rules is shown in figure 21. Note that the minimum width for a diffused region applies to diffused regions formed between a normal boundary

Fig.15. $W_d/\lambda \geq 2$

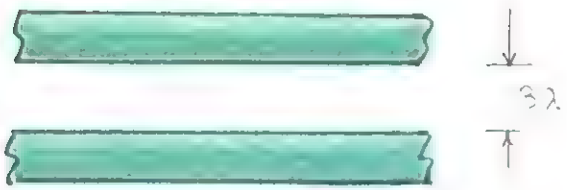


Fig.16. $S_{dd}/\lambda \geq 3$

Fig.17. $W_p/\lambda \geq 2$

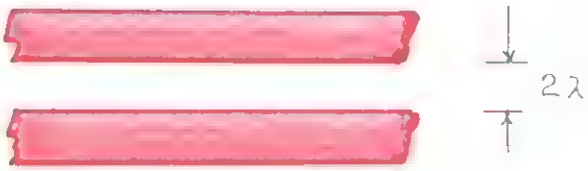


Fig.18. $S_{pp}/\lambda \geq 2$

Fig.19. $S_{pd}/\lambda \geq 1$

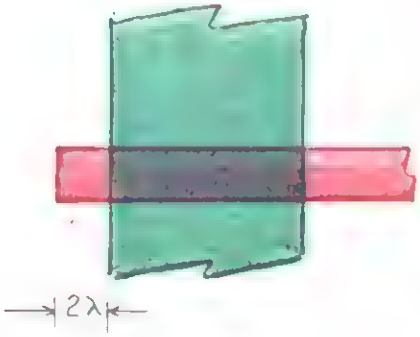


Fig.20. $E_{pd}/\lambda \geq 2$

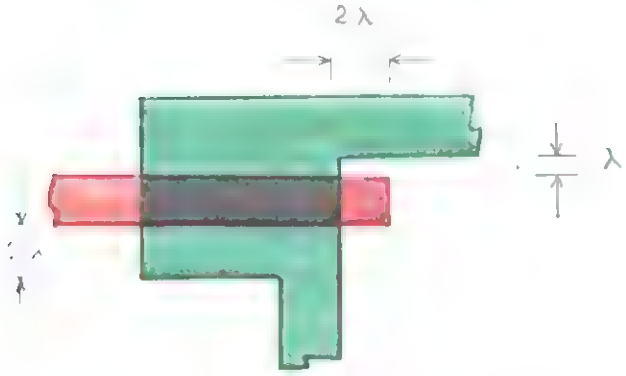


Fig.21. Example of Several Rules

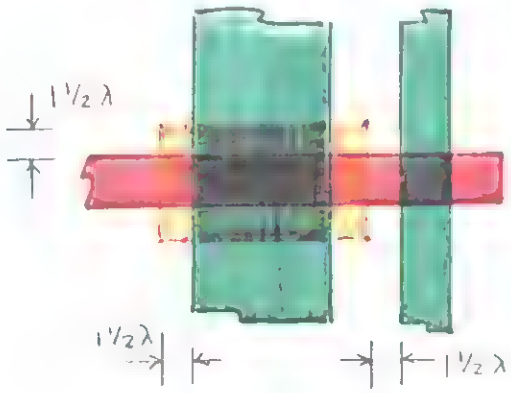


Fig.22. $S_{ig}/\lambda \geq 1\frac{1}{2}$ $E_{ig}/\lambda \geq 1\frac{1}{2}$

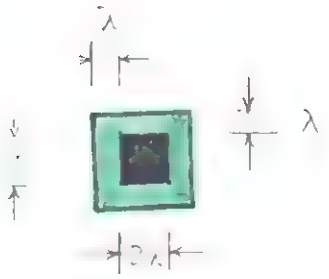


Fig.23 $W_c/\lambda \geq 2$, $F_{dc}/\lambda \geq 1$

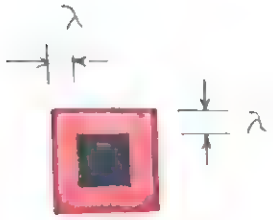


Fig.24. $F_{pc}/\lambda \geq 1$

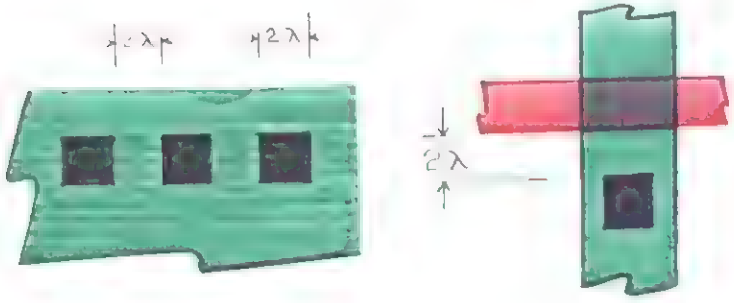


Fig.25. $S_{cc}/\lambda \geq 2$, $S_{cg}/\lambda \geq 2$

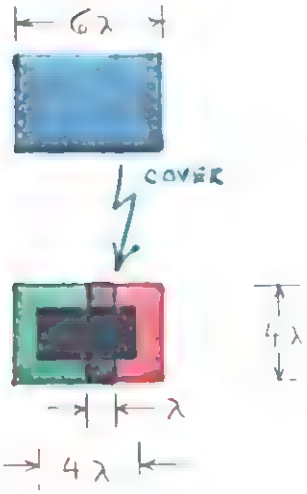


Fig. 26. $O_{pd}/\lambda = 1$, and details of butting contact

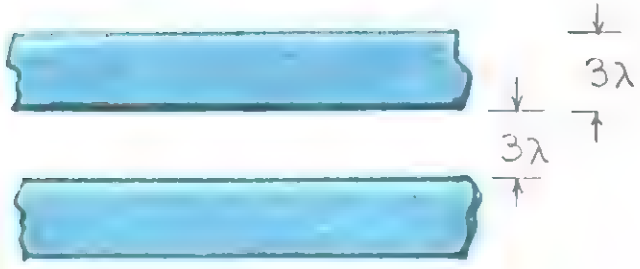


Fig.27. $W_m/\lambda \geq 3$, $S_{mm}/\lambda \geq 3$

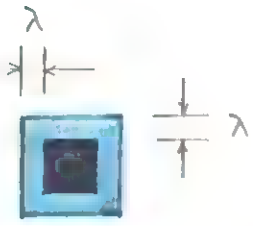


Fig.28. $F_{mc}/\lambda \geq 1$

of the diffused region and an edge of a transistor as well as to a diffused line formed by two normal boundaries. This situation is illustrated in the lower left corner of the figure.

As we have seen in figure 10, ion implantation in the region which becomes the gate of a transistor will convert the resulting transistor into the depletion mode type. It is important that the implanted region extend outward beyond all four boundaries of the gate region, as shown in figure 22. To avoid any possibility that some small fraction of the transistor might remain enhancement mode, the yellow ion implantation region should extend a distance E_{ig} of at least 1½ λ beyond each edge of the gate region. The separation S_{ig} between an ion implantation region and an adjacent enhancement mode transistor gate region should also be at least 1½ λ. Both situations and their design rules are illustrated in the figure.

A contact may be formed between the metal layer and either the diffused level or the polysilicon level by means of the contact mask. A set of rules apply to the amount by which each layer must provide an area surrounding any contact to it, so that the contact opening not find its way around the layer to something unintended below it. Since no physical factors apply here other than the relative registration of two levels, a very simple set of design rules results. Each level involved in a given contact must extend beyond the outer boundary of the contact cut by λ at all points, as illustrated by extension distances E_{dc}, E_{pc}, and E_{mc}, in figures 23, 24, and 26. The contacts themselves, like the minimum width lines in the other levels, must be at least 2λ long and 2λ wide (W_c). This situation is illustrated for the diffusion and polysilicon levels in figures 23 and 24. When making contact between a large metal region and a large diffused region, many small contacts spaced 2 λ apart should be used, as shown in figure 25. Contact cuts to diffusion should be at least 2λ from the nearest gate region, as shown in figure 25.

A special case arises when it is desired to connect a polysilicon layer directly to a diffused layer, using the butting contact. The detailed geometric layout of the butting contact is shown in figure 26. In its minimum sized configuration, it is composed of a square region of diffision 4 λ on a side, overlapped by a 3 λ by 4 λ rectangle of polysilicon. A rectangular contact cut, 2 λ by 4 λ in size, is made in the center of this structure. The structure is then overlayed with metal, thus connecting the polysilicon to the diffusion. The rules involved in figure 26 are identical to those given so far, with the addition of a minimum of one λ overlap, O_{pd}, of the diffused and polysilicon layers in the center area of the contact.

In considering the design rules for the metal layer, notice that this layer in general runs over much more rugged terrain than any other level, as can be seen by referring to the cross section of figure 14. For this reason it is generally accepted practice to allow somewhat wider minimum lines and spaces for the metal layer than for the other layers. As a good working rule 3λ widths (W_m), and 3λ separations (S_{mm}) between independent metal lines should be provided, as shown in figure 27. The metal layer must surround the contact layer in much the same way that the diffused and polysilicon layers did. Since the resist material used for patterning the metal generally accumulates in the low areas of the wafer, it tends to be thicker in the neighborhood of contact than elsewhere. For this reason metal tends to be slightly larger after patterning in the vicinity of a contact than elsewhere. It is generally sufficient to allow only one λ of space around the contact region for the metal, as for the other two layers. The rule for metal surrounding contacts is shown in figure 28.

The above design rules are likely to remain valid as the length-unit λ scales down in size with the passage of time. Occasionally, for specific commercial fabrication processes, some one or more of these rules may be relaxed or replaced by more complex rules, enabling slight reductions in the area of a system. While these details may be important for certain competitive products such as memory systems, they have the disadvantgage of making the system design a captive of the process specific design rules. Extensive redesign and checking is required to scale down such a design as the length-unit scales down. For this reason, we recommend use of the dimensionless rules given, especially for prototype systems. Designs implemented according to these rules are easily scaled, and may have reasonable longevity.

Formal Description of Design Rules

{ in preparation }

Electrical Parameters

By satisfying the constraints imposed by the design rules, designers may create circuit layout patterns with the knowledge that the appropriate transistors, lines, etc., produced by the wafer fabrication process will be as originally specified in their layout patterns. To complete a design it is necessary to also know the electrical parameters of the transistors, diffused layers, polysilicon layers, etc., so that the performance of circuits can be evaluated. The resistances per square of the various layers and the capacitance per square micron with respect to underlying substrate are shown in Table 1. Note that the resistance of a square of material contacted along two opposite sides is independant of the size of the square, and equals the resistivity of the material divided by its thickness. The tabulated values are typical of processes running in 1978. As the circuit dimensions are scaled *down* by dividing by a factor α, the parameters scale approximately as shown in the table.

Resistances:	Metal	~ 0.1 ohms/□	Resistances/square scale *up* by α.,
	Diffusion	~ 10 ohms/□	as dimensions scale *down* by α,
	Poly	~ 15-100 ohms/□	except that the transistor
	Transistor	~ 10^4 ohms/□	R/□ is independent of α
Capacitances:	Gate-channel	~ 4×10^{-4} pf/μm^2	Capacitances/micron^2 scale
	Diffusion	~ 0.8×10^{-4} pf/μm^2	*up* by α., as dimensions
	Poly	~ 0.4×10^{-4} pf/μm^2	scale *down* by α
	Metal	~ 0.3×10^{-4} pf/μm^2	

Table 1. Typical MOS Electrical Parameters (1978).

The relative resistance values of metal, diffusion, poly, and drain to source paths of transistors are quite different. Diffusion and good polysilicon layers have approximately one hundred times the resistance per square area of the metal layer. A fully turned on transistor has approximately one thousand times the resistance of the diffused and polysilicon layers. The capacitances are not as wildly different as the resistances of the various layers. Compare the capacitances in Table 1 to the gate to channel capacitance, as a reference. The diffused areas typically have one fifth the capacitance per square micron. Polysilicon on thick oxide has approximately one tenth, and the metal layer slightly less than one tenth, of the gate-channel capacitance per square micron.

The relative values of the resistances and capacitances are not expected to vary dramatically as the processes evolve towards smaller dimensions, with the exception of the transistor resistance per square, which is independent of α. One note of warning: There is a wide range of possible values of polysilicon resistance for different commercial processes. Polycrystalline silicon suffers from inordinately high resistances at the crystal grain boundaries if the doping level in the polysilicon itself is not held quite high. This disease does not affect the diffused layers. For this reason, any processing which tends to degrade the doping levels in the diffused and polysilicon layers, affects the polysilicon resistance much more dramatically than the resistance of the diffused area. It is in general difficult to design circuits which are optimum over the entire range of polysilicon resistivity. If a circuit is to be run on a variety of fabrication lines, it is desirable for the circuit to be designed in such a way that no appreciable current is drawn through a long thin line of polysilicon. In an important example in Chapter 5., polysilicon lines are used as buses along which information flows. The timing of these buses can be dramatically affected by the resistance of the polysilicon. However, the protocol used on these busses has the polysilicon lines precharged during one period of a clock and then pulled low by the appropriate bus source during a following clock period. In this way the circuit is guaranteed to work independent of the resistance of the poly. However, it may be considerably slower in processes of high poly resistivity.

Current Limitations in Conductors

One limit which is not covered in either the design rules or the electrical parameters section is that associated with the maximum currents through metal conductors. There is a physical process called *metal migration* whereby a current flux through a metal conductor, exceeding a certain limit, causes the metal atoms to move slowly in the direction of the current. If there is a small constriction in the metal, the current density will be higher and therefore more metal atoms will be carried forward from that point, narrowing the point still more. Hence, metal migration is a destructive mechanism causing open circuits in the metal layer carrying heavy currents.

For metals like aluminum this limit is a few times 10^5 amperes per square centimeter. This limit does not interfere too drastically with the design of integrated systems in current MOS technologies. However, many metal conductors in present integrated systems are operated near their current limit, and currents do not scale well as the individual elements are made smaller. Applying the scaling rules developed earlier, we found that the power per unit area is independent of the scale down ratio. However, the supply voltage decreases and therefore the current per unit area increases as the devices are scaled down. For this reason it will not be possible to use processes for very large scale integrated systems where the metal thickness scales in the same way as other dimensions in the circuit. Much work will likely be done to develop processes enabling fabrication of metal lines of greater height relative to width than is presently possible.

Short pulses of current are known to contribute much less to metal migration than steady direct current. Nanosecond pulses of currents two orders of magnitude higher than the dc limit given above may be carried in metal conductors without apparent damage. Therefore, switching current may not be as damaging to metal conductors as a steady current.

These effects strongly favor processes like CMOS which do not require static dc current, and favor design methodologies which maximize system function per unit dc current.

A Closer Look at Some Details

Thus far our discusssion of fabrication has been a general one, adequate for readers whose primary interest is in the systems aspects of VLSI. The following sections involve a more detailed examination of the capacitance of several important structures and a discussion of the relative merits and scaling behavior of several common processes. We suggest that the reader just skim through these sections during the first reading of this text.

In Table 1 we gave typical capacitance for the various layers to the substrate. These capacitances are those which would be measured if the voltage on the particular layer were zero (relative to the substrate). The dependence upon voltage of the capacitance of the different layers may sometimes be important and we will now discuss how this dependence arises. References R1, R2, R4, and Reference 4 of Chapter 1, are good sources for those wishing more background information on the concepts of device physics used in this text.

When a negative voltage is applied to an n-type diffused region relative to the p-type bulk silicon, the negative electrons are pushed out of the n-type layer into the bulk and a current flows. In integrated systems we are careful to never allow the voltage on the n-type diffused regions to be more negative than the p-type bulk. Diffused regions are biased positively with respect to the p-type bulk, resulting in a reversed biased p/n junction. With the exception of a small leakage current, the reverse biased p/n junction acts merely to isolate one diffused region from another. The p-type bulk of our integrated system has a small number (typically 10^{15}-10^{16} per cubic centimeter) of impurity atoms. When a voltage is applied to an n-type diffused region, its influence is felt well out into the p-type bulk. Positive charge carriers in the p-type bulk are repelled from the positively charged n-type layer, thereby exposing negatively charged impurity ions. The region surrounding the n-type diffused layer which has been depleted of positive charge carriers is referred to as a depletion layer and is shown schematically in Figure 29b. As the voltage on the n-type layer is increased, charge carriers are pushed further back from the junction between the n-type layer into the p-type bulk, widening the depletion layer and exposing more charged impurity ions. The charge thus induced in the depletion layer as the voltage on the n-type diffused region is increased is responsible for the capacitance of the n-type diffused region relative to the substrate.

We will now consider a unit area of the junction. The total charge in the depletion layer per unit area is proportional to the number per unit volume of impurity ions in the bulk

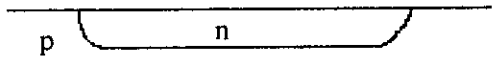


Fig. 29a. n-type Diffusion in p-type Bulk Silicon

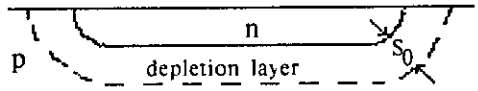


Fig. 29b. Depletion Layer

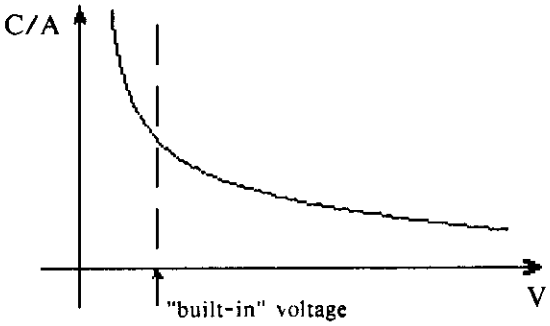


Fig. 30. C/A as fcn(V)

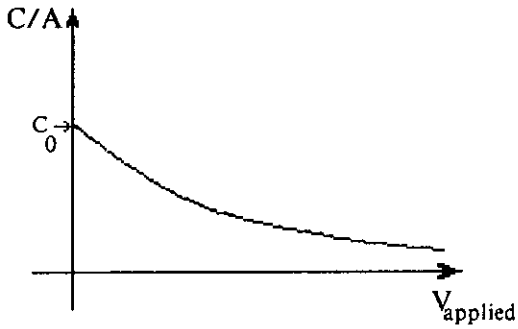


Fig. 31. C/A as fcn ($V_{applied}$)

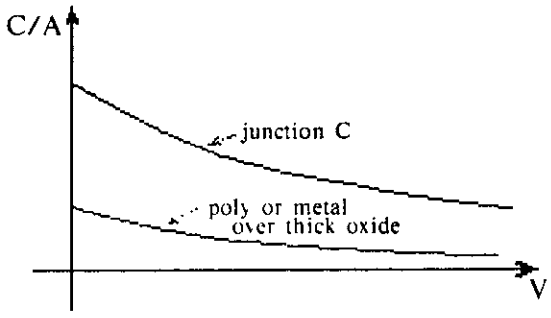


Fig. 32. Capacitance of Poly or Metal over Thick Oxide

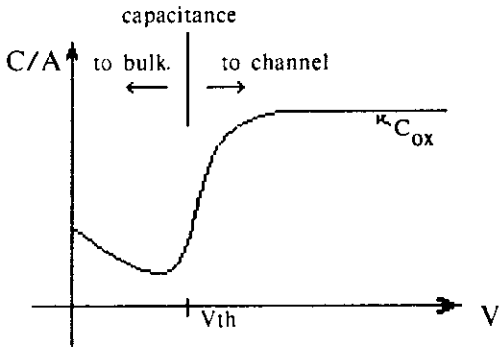


Fig. 33. MOS Gate Capacitance as fcn(V)

(graphs.press)

(N), and the width, s_0, of the depletion layer.

$$\text{Total charge/area} \propto Ns_0$$

The electric field in the region is proportional to the charge per unit area.

$$\text{Electric field} \propto \text{charge/area} \propto Ns_0$$

The voltage between the n-type diffused layer and the p-type bulk on the far side of the depletion layer is proportional to the electric field times thickness of the depletion layer, and therefore to the density of negatively charged ions in the depletion layer times the square of the width of the depletion layer.

$$\text{Voltage} \propto \text{electric field} \times s_0 \propto Ns_0^2$$

The capacitance per unit area is just the charge per unit area divided by the voltage across the depletion layer. From the above equations the capacitance is proportional to the square root of the density of impurity atoms in the p-type bulk divided by the voltage.

$$\text{Capacitance/area} = Q/V \propto 1/s_0 \propto (N/V)^{1/2}$$

This relationship is plotted in Figure 30. Notice that the capacitance tends towards infinity as the voltage across the junction tends to zero. It would seem that this large capacitance would be disastrous for the performance of our integrated systems. However, this proves not to be the case. When the p/n junction was formed the n-type region had an excess of negative charge carriers while the p-type bulk had an excess of positive charge carriers. When the two were brought together to form the junction, there was no voltage to prevent charge carriers of either type from flowing over into the opposite region. This initial flow caused the n-type layer to become more positive than the p-type layer. This flow ceased when just enough voltage built up to stop it. In silicon the voltage required to prevent the flow of charge carriers in such a situation is approximately 0.7 volt. Thus the true voltage across the junction is this initial "built-in" voltage plus the voltage we apply in our circuit. The variation of the capacitance per unit area with applied voltage is shown in Figure 31. An approximate equation which can be used to calculate the junction capacitance C_j per unit area of diffused layers as a function of the applied voltage is given by:

$$C_j = 4.5\times10^{-12}[N/(V+0.7)]^{1/2} \text{ pf}/\mu m^2$$

In this equation, N, the density of impurity ions in the p-type bulk, should be given in number per cm^3. The voltage is in volts and the capacitance per unit area is evaluated in picofarads per square micron. This equation is adequate for most design purposes.

Aside from the diffused regions, there are two other situations where the capacitance is of interest. The first is poly or metal over thick oxide and the second is the gate of an MOS transistor. We will discuss poly or metal over oxide first. Figure 32 illustrates once more the capacitance per unit area of a junction over the p-type bulk. If the poly or metal layer was lain on an oxide much thinner than the depletion layer, its capacitance would be nearly the same as that of the corresponding p/n junction. However, if an oxide is interposed whose thickness is of the order of the depletion layer thickness, the capacitance of the poly or metal line will be decreased. The formula which applies in this case is given by:

$$1/C_{total} = 1/C_j + 1/C_{ox}$$

A typical dependence is shown in Figure 32. For an oxide thickness d, $C_{ox} = 3.5\times10^{-2}/d$, where the thickness d is given in angstrom units (10^{-4} microns), and the result is in picofarads per square micron as before.

The most spectacular voltage dependence of a capacitance in the technology we will be using is that of the gate of an MOS transistor. When the gate voltage V_{gs} is less than the threshold voltage V_{th}, the capacitance of the gate to the bulk is just that given above for metal or poly over oxide since the voltage on the gate merely depletes positive charge carriers back from the channel area. However, when the voltage on the gate reaches the threshold voltage of the transistor, negative charge is brought in under the gate oxide from the source of the transistor and the capacitance changes abruptly from the small valve associated with depleting charges in the bulk to the much larger oxide capacitance between the gate and the channel region. Further increase in voltage on the gate merely increases the amount of mobile charge under the gate oxide with no change in the width of the depletion layer underneath the channel. Hence, the character of the gate capacitance changes abruptly as the gate voltage passes through the threshold voltage.

The dependence of the total gate capacitance on gate voltage is shown in figure 33. The capacitance from channel to bulk is completely separate from the gate to channel

capacitance. It is associated with the depletion layer underneath the channel region, and is almost identical to that of a diffused region of the same area. When the gate voltage is below threshold, the gate to channel capacitance disappears altogether leaving only the small parasitic overlap capacitances between the gate and the source and drain regions.

Choice of Technology

Before proceeding to the chapters on system design, let us briefly examine some alternative technologies. Using the knowledge developed in these first two chapters, we will discuss the reasons for selecting nMOS as the single technology used to illustrate integrated systems in this text. Some of the factors which must be considered in choosing a technology include circuit density, richness of available circuit functions, performance per unit power, the topological properties of circuit interconnection paths, suitablility for total system implementation, and general availability of processing facilities.

As the technology advances, more system modules can be placed on the same sized chip. An ultimate goal is the fabrication of large scale systems on single chips of silicon. For this goal to be attained, any signal which is required in the system other than inputs, outputs, VDD, and GND, must be generated in the technology on the chip. In other words, no subsystem can require a different technology for the generation of its internal signals. Thus such technologies as magnetic bubbles are ruled out for full integrated systems because they are not able to create the signals required for all operations in the on-chip medium.

We believe that for any silicon technology to implement practical large scale systems, it must provide two kinds of transistor. The rationale for this observation is as follows. In order to provide some kind of nonlinear threshold phenomenon there must be a transistor which is normally off when its control input is at the lowest voltage used in the system. Bipolar technologies use NPN transistors for this purpose. The nMOS technology uses n-channel enhancement mode devices. In addition to this transistor, a separate type of transistor must be supplied to allow the output of a driver device to reach the highest voltage in the circuit (VDD). In the bipolar technologies, PNP lateral devices are used to supply this function, in the n-channel technology a depletion mode device is used, and in complementary MOS technology a p-channel enhancement mode device is used. All three choices allow output voltages of drivers to reach VDD and thus meet the above criterion.

To date three technologies have emerged which are reasonably high in density and scale to submicron dimensions without an explosion in the power per unit area required for their operation. These are the n-channel silicon gate process, the complementary MOS silicon gate process, and the integrated injection logic, or I^2L, process[2]. Although present forms of I^2L technology lack the additional level of interconnect available in the silicon gate technologies, there is no inherent reason that such a level could not be provided. It is important to note that increasing the flexibility of interconnect enriches the types of array functions which can be created. I^2L has the advantage over nMOS that the power per unit area (and hence the effective τ of its elementary logic functions) can be controlled by an off chip voltage. The decision concerning what point on the speed vs power curve to operate may thus be postponed until the time of application (or even changed dynamically).

The nMOS scaling has been described previously. Any technology in which a capacitive layer on the surface induces a charge in transit under it to form the current control "transistor" will scale in the same way. Examples include Schottky Barrier Gate FET's (MESFETs), Junction FET's, and CMOS.

There are certain MOS processes (VMOS, DMOS) of an intermediate form in which the channel length is determined by diffusion profiles. While competitive at present feature sizes, these are likely to be interim technologies which will present no particular advantage at submicron feature sizes.

Scaling of the bipolar technology[1] is quite different from that of MOS technologies. For completeness, we include here a discussion of the scaling of bipolar devices, which may be of interest to those familiar with those technologies.

Traditionally, bipolar circuits have been "fast" because their transit time was determined by the narrow base width of the bipolar devices. In the 1950's, technologists learned how to form bipolar transistor base regions as the difference between two impurity diffusion profiles. This technique allowed very precise control of the distance perpendicular to the silicon surface, and therefore permitted the construction of very thin base regions with correspondingly short transit times. Since current in a bipolar device flows perpendicular to the surface, both the current and the capacitance of such devices are decreased by the same factor as the device surface dimensions are scaled down, resulting in no change in time performance. The base widths of high performance bipolar devices are already nearly as

thin as device physics allows. For this reason, the delay times of bipolar circuits is expected to remain approximately constant as their surface dimensions are scaled down.

The properties of bipolar devices may be analyzed as follows. The collector current is due to the diffusion of electrons from emitter to collector. For a minority carrier density N(x) varying linearly with distance x, from N_0 at the emitter to zero at the collector (at x=d), the current I per unit area A is:

$$I/A = q(2D)dN/dx = q(2D)N_0/d = q(2kT/q)\mu N_0/d \qquad [eq.2]$$

where the diffusion constant $D = \mu kT/q$. The factor of two multiplies the diffusion constant in eq.2 because high performance bipolar devices operate at high injection level (injected minority carrier density much greater than equilibrium majority carrier density). The inherent stored charge in the base region is:

$$Q/A = N_0 d/2 \qquad [eq.3]$$

Therefore, the transit time is:

$$\tau = Q/I = d^2/[4\mu kT/q] \qquad [eq.4]$$

The form of equation 4 is exactly the same as that for MOS devices (eq.1., chapter 1.), with the voltage in the bipolar case being equal to 4kT/q (at room temperature kT/q = 0.025 volts). A direct comparison of the transit times is shown in Table 2.

Table 2. Transit Time:
τ = (Distance)^2/(Mobility×Voltage)

	MOSFET:	MESFET, JFET:	Bipolar:
Distance:	channel length	channel length	base width
Voltage:	~ VDD/2, many kT/q	~ VDD/2, many kT/q	4kT/q
Mobility: $cm^2/v\text{-}sec$, (Si)	surface mobility, ~800	bulk mobility, ~1300	bulk mobility, ~1300

At the smallest dimensions to which devices can be scaled, the base width of bipolar devices and the channel length of FET devices are limited by the same basic set of physical constraints, and are therefore similar in dimension. The voltage on the FET devices must be many times kT/q to achieve the required nonlinearity. Hence at ultimately limiting small dimensions the two types of device have roughly equivalent transit times. At these limiting dimensions, choices between competing technologies will be made primarily on the grounds of the topological properties of their interconnects, the functional richness of their basic circuits, simplicity of process, and ability to control dc current per unit area. As supply voltages are scaled down to the 1 volt range, MOS devices become similar in most respects to other FET type devices, and it is possible that mixed forms (MOS-JFET, MOS-MESFET, Bipolar-MESFET, etc.) may emerge as the ultimate integrated system technologies.

We have chosen to illustrate this text with examples drawn from the n-channel silicon gate depletion mode load technology. The reasons for this choice in 1978 are quite clear. In addition to meeting the required technical criteria we have described, this technology provides some important practical advantages to the student and to the teacher. It is the only high density technology which has achieved universal acceptance across company and product boundaries. Readers wishing to implement integrated system designs may have wafers fabricated by essentially any wafer fabrication firm, without fear that slight changes in the process or the vagaries of relationships with a particular firm will cut off their source of supply. It is also presently the highest density process available. This certainty of access to fabrication lines, the more generally widespread knowledge of nMOS technology among members of the technical community, its density, and its performance similarity with bipolar technology in its ultimate scaling, are all important factors supporting its choice for this text on VLSI Systems. However, the principles and techniques developed in this text can be applied to essentially any technology.

References

1. B. Hoeneisen, C. A. Mead, "Fundamental Limits in Micro-electronics--II. Bipolar Technology", *Solid-State Electronics*, Vol.15, 1972, pp. 891-897.

2. F. M. Klaassen, "Device Physics of Integrated Injection Logic", *IEEE Transactions on Electron Devices*, March 1975, pp. 145- , and cited papers by Hart & Slob, and by Berger & Weidmann.

Reading References

R1. A. S. Grove, "Physics and Technology of Semiconductor Devices", J. Wiley and Sons, 1967, is the early classic text on process technology and device physics.

R2. W. G. Oldham, "The Fabrication of Microelectronic Circuits", *Scientific American*, September, 1977, provides an excellent overview of the fabrication process.

R3. I. E. Sutherland, C. A. Mead, T. E. Everhart, "Basic Limitations in Microcircuit Fabrication Technology", ARPA Report R-1956-ARPA, November, 1976, contains a quantitative discussion of the many limiting factors in fabrication.

R4. R. S. Muller, T. I. Kamin, "Device Electronics for Integrated Circuits", Wiley, 1977, provides insight into the device physics relevant to current integrated circuit practice.

Chapter 3: Data and Control Flow in Systematic Structures

Sections:

The process of designing a large-scale integrated system is sufficiently complex that only by adopting some type of regular, structured design methodology can one have hope that the resulting system will function correctly, and not require a large number of redesign iterations. However, the methodology used should allow the designer to take full advantage of the architectural possibilities offered by the underlying technology.

In this chapter we present a number of examples of data and control flow in regularized structures, and discuss the way in which these structures can be assembled into larger groups to form subsystems, and then these subsystems assembled to form the overall system. The design methodology suggested in this chapter is but one of many ways in which integrated system design may be structured. The particular circuit form presented does tend to produce systems of very simple and regular interconnection topology, and thus tends to minimize the areas required to implement system functions. Arrays of pass transistor logic in register to register transfer paths are used wherever possible to implement system functions. This approach tends to minimize power dissipated per unit area, and, with level restoration at appropriate intervals, tends to minimize the time delay per function. The methodology developed is applied in later chapters to the architecture and design of a data processing path and its controller, which together form a microprogrammed digital computer.

Computer architects, who usually design systems in a rather structured way using commercially available MSI and LSI circuit modules, are often surprised to discover how unstructured is the design within those modules. In principle one can use the basic NAND and NOR logic gates described in Chapter 1 to implement combinational logic, build latches from these gates to implement data storage registers, and then proceed to design integrated systems using traditional logic design methodology as applied to discrete devices. Integrated systems are often designed this way at the present time. However, it is unlikely that such unstructured approaches to system design can survive as the technology scales down towards maximum density VLSI.

There are historical reasons for the extensive use of random logic within integrated systems. The first microprocessors produced by the semiconductor industry were fairly direct mappings of early generation central processor architectures into LSI. A block diagram of the Intel 4004, the earliest microprocessor to see widespread commercial application, is illustrated in figure 1a. The actual chip layout of the 4004 shown in Figure 1b indicates the complexity of the LSI implementation of this simple central processing unit. Such LSI systems, directly mapping data paths and control functions appropriate in earlier component technologies, of necessity contained a great deal of random logic. However, the extensive use of random logic results in chip designs of very great geometrical and topological complexity, relative to their logical processing power.

To deal with such complexity, system design groups have often stratified the design problem into architecture, logic design, circuit design, and finally circuit layout, with specialists performing each of these levels of the design. Such stratification often precludes important simplifications in the realization of system functions.

Switching theory provides formal methods for minimizing the number of gates required to implement logic functions. Unfortunately, such methods are of little value in VLSI systems, since the area occupied on the silicon surface by circuitry is far more a function of the topological properties of the circuit interconnections than it is of the number of logic gates implemented. The minimum gate implementation of a function often requires much more surface area for its layout than does an alternative design using more transistors but having simpler interconnection topology.

There are known ways of structuring integrated circuit designs implemented using traditional logic design methods. A notable example is the *polycell*[1] technique. In this technique, a group of standard cells corresponding to typical SSI or MSI functions are gathered into a library of functions. The logic diagram for the system to be implemented is used to specify which cells in the library are required. The cells are then placed into a chip layout, and interconnections laid out between them by an automatic interconection routing system. The polycell technique provides the logic designer having limited knowledge of integrated systems with a means of implementing modest integrated circuit designs directly from logic equations. However, a heavy penalty is paid in area, power, and delay time. Such techniques, while valuable expedients, do not take advantage of the true architectural potential of the technology, and do not provide insight into directions for further progress.

The Intel 4004 Microprocessor: An Early LSI System

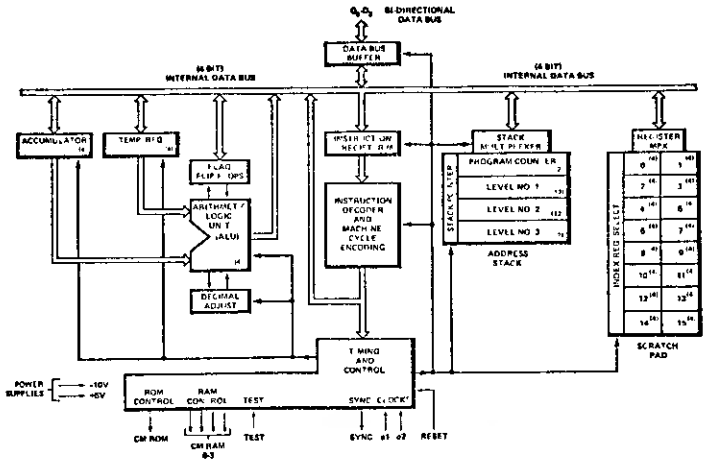


Fig. 1a. 4004 Block Diagram

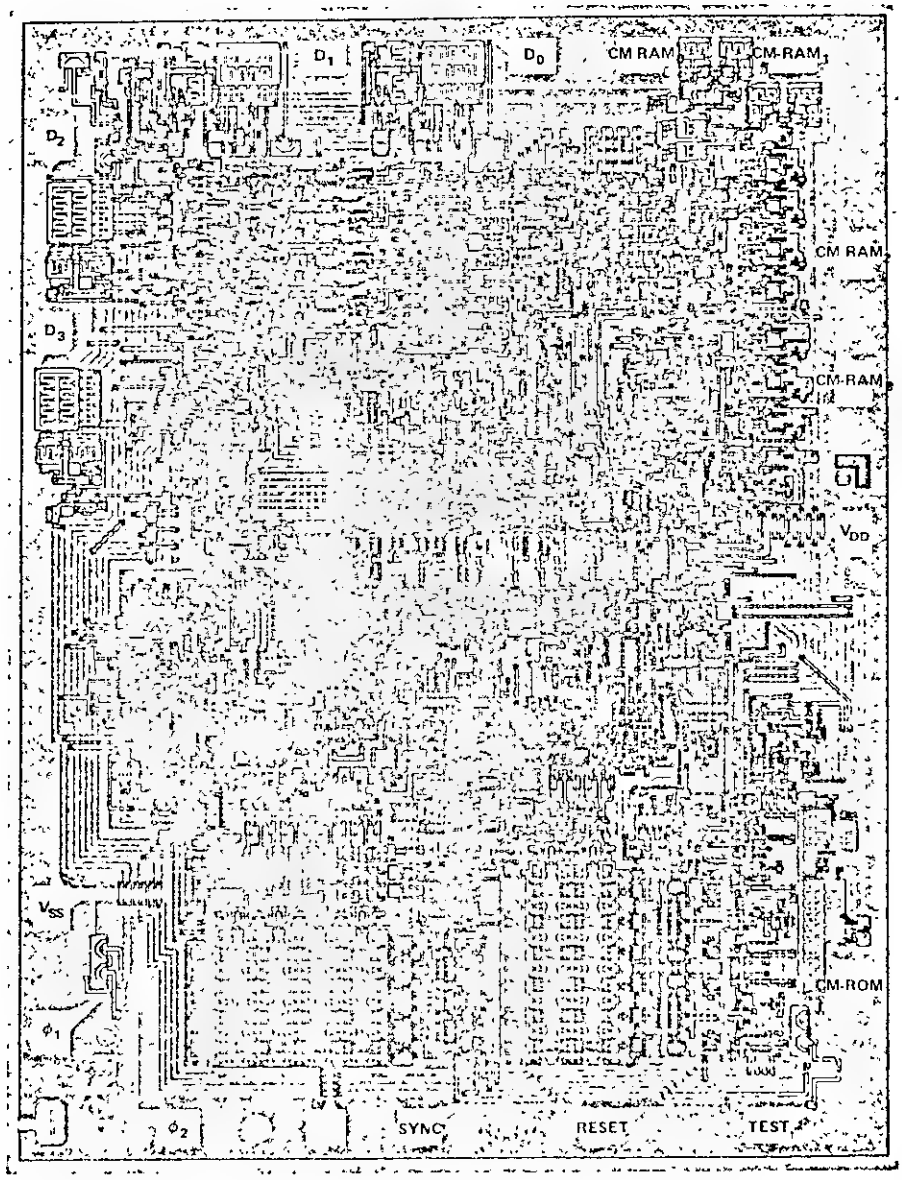


Fig 1b. 4004 Chip Photomicrograph with Pin Designations

[reprinted with permission of G. Moore, Intel Corp.]

Switching theory not only yields the minimum number of gates to implement a logic function, but it also directly synthesizes the logic circuit design. Unfortunately, at the present time there is no general theory which provides us with a lower bound on area, power, and delay time for the implementation of logic functions in integrated systems. Theoretical lower bounds for certain special structures and algorithms of interest are given in chapter 9.

In the absence of a formal theory, we can at best develop and illustrate alternative design methodologies which tend to minimize these physical parameters. Proposed design methodologies should in addition provide means of structuring system designs so as to constrain complexity as circuit density increases. We hope that the examples and techniques presented in this text will serve to clarify these issues and stimulate others to join in the search for more definitive results[2].

Notation

There are a number of different levels of symbolic representation for MOS circuits and subsystems used in this text. Figures 2a., 2b., 2c., and 2d., illustrate a NAND gate at several such levels. At times it may be necessary to show all the details of a circuit's *layout geometry* in order to make some particular point. For example, a clever variation in some detail of a circuit's layout geometry may lead to a significant compaction of the circuit's area without violating the design rules.

Often, however, a diagram of just the topology of the circuit conveys almost as much information as a detailed layout. Such *stick diagrams* may be annotated with important circuit parameters if needed, such as the L/W ratios shown in figure 2b. Many of the important architectural parameters of circuits and subsystems are a reflection of their interconnection topologies.

Alternative topologies often lead to very different layout areas after compaction. The discovery of a clever starting topology for a design usually provides far better results than does the application of brute force to the compression of final layout geometries. For this reason, many of the important structural concepts in this chapter and throughout the text will be represented for clarity by use of colored *stick diagrams*. The color coding of the

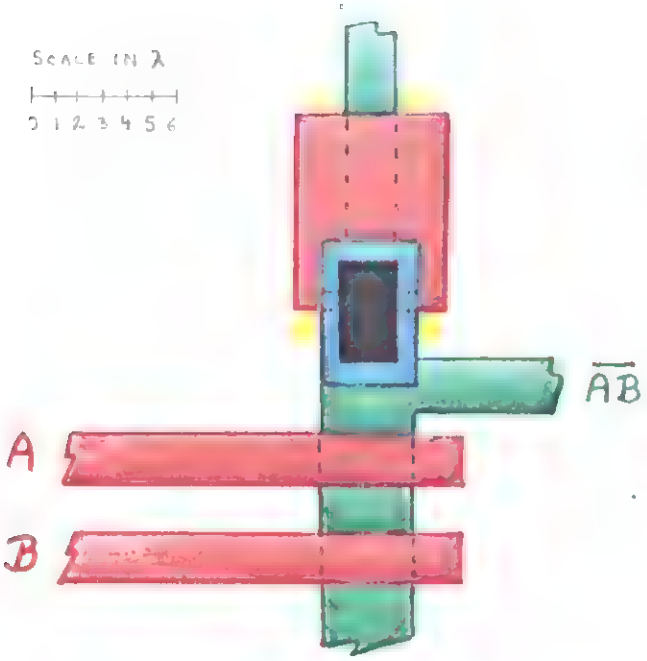


Fig.2a. NAND Gate: Layout Geometry

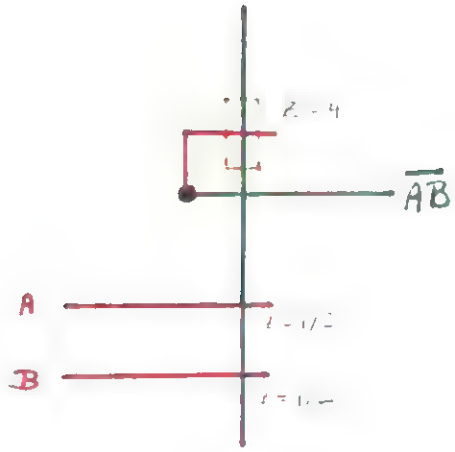


Fig.2b. NAND Gate: Topology
(Stick Diagram)

Fig.2c. NAND Gate: Circuit Diagram

Fig.2d. NAND Gate: Logic Symbol

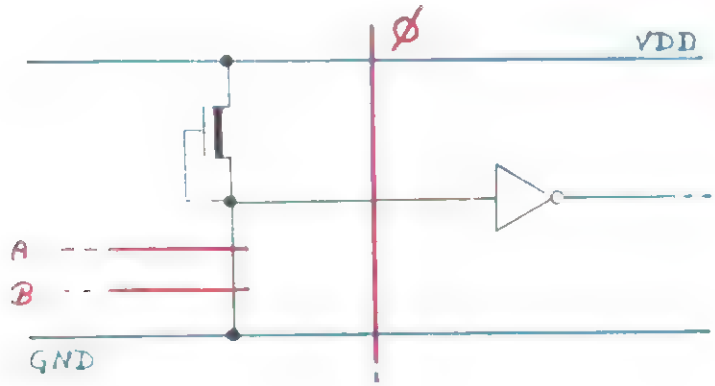


Fig. 2e. Example of Mixed Notation

stick diagrams is the same as that of layout geometries, and is as follows: *green* symbolizes *diffusion*; *yellow* symbolizes *ion implantation* for depletion mode transistors; *red* symbolizes *polysilicon*; *blue* symbolizes *metal*; *black* symbolizes a *contact*.

Later, through a number of examples in chapter 4, we will present the details of procedures by which the stick diagrams are transformed into circuit layouts, and then digitized for maskmaking. Note that if this topological form of representation were formalized, one might consider "compiling" such descriptions by implementing algorithms which "flesh out and compress" the stick diagrams into the final layout geometries[3], according to the constraints imposed by the design rules.

When the details of neither geometry nor topology are needed in the representation, we may revert to the familiar *circuit diagrams* and *logic symbols*. At times we may find it convenient to *mix* several levels in one diagram, as shown in figure 2e. A commonly used mixture is: (i) stick diagrams in portions where topological properties are to be illustrated, (ii) circuit symbols for pullups, and (iii) logic symbols, or defined higher level symbols, for the remaining portions of the circuit or system.

We will define logic variables in such a way that a *high voltage* on a signal path representing that variable corresponds to that variable being *true* (logic-1). Conversely, a *low voltage* on a signal path representing that logic variable corresponds to the variable being *false* (logic-0). Here *high voltage* and *low voltage* mean well above and well below the logic threshold of any logic gates into which the signal is an input. This convention simplifies certain discussions of logic variables and the voltages on the signal paths representing them. Thus when we refer to the logic variable β being *high*, we indicate simultaneously that β is *true* (logic-1) and is represented on the signal path named β by a *high voltage*, one well above the logic threshold. In boolean equations and logic truth tables we use the common notation of 1 and 0 to represent *true* and *false* respectively, and by implication *high* and *low voltages* on corresponding signal paths.

Two Phase Clocks

We will often make use of a particular form of "clocking" scheme to control the movement of data through MOS circuit and subsystem structures. By clocking scheme we mean a strategy for defining the times during which data is allowed to move into and through successive processing stages in a system, and for defining the intervening times during which the stages are isolated from one another. Many alternative clocking schemes are possible, and a variety are in current use in different integrated systems[4]. The clocking scheme used in an integrated system is closely coupled with the basic circuit and subsystem structuring, and has major architectural implications. For clarity and simplicity we have selected one clocking scheme, namely *two-phase, non-overlapping clock signals.* This scheme is used consistently throughout the text, and is well matched to the type of basic structures possible in MOS technology.

The two clock signals φ_1 and φ_2 are plotted as a function of time in figure 3. The signals both switch between zero volts (logic-0) and a voltage near VDD (logic-1), and both have the same period, T. Note that both signals are non-symmetric, and have non-overlapping *high* times. The *high* times are somewhat shorter than the *low* times. Thus φ_2 is *low* all during each of those time intervals from when φ_1 rises, nears VDD, and then falls back to zero. Symbolizing the *rise* of a signal φ as $\uparrow\varphi$, and the *fall* as $\downarrow\varphi$, we also have a similar rule for φ_1, namely $\varphi_1 = 0$ all during each time interval from $\uparrow\varphi_2$ to $\downarrow\varphi_2$. Therefore, at all times the logic AND of the two signals equals zero: $[\varphi_1(t)] \cdot [\varphi_2(t)] = 0$, for all t. For convenience, we will often use the following equivalence in our descriptions: *"during the time period when φ_i is high"* $\equiv$ *"during φ_i"*. In the next section we will illustrate the use of these two clocking signals to move data through some simple MOS circuit structures.

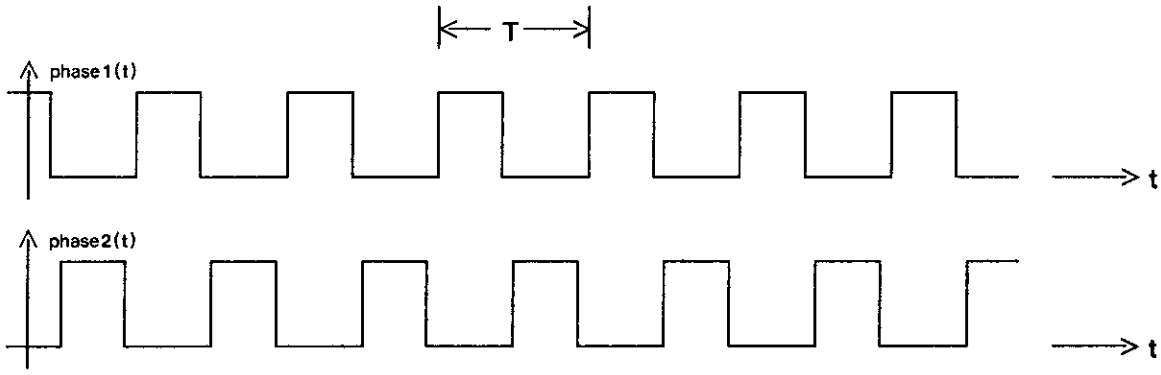


Fig. 3. Two Phase Non-Overlapping Clock Signals

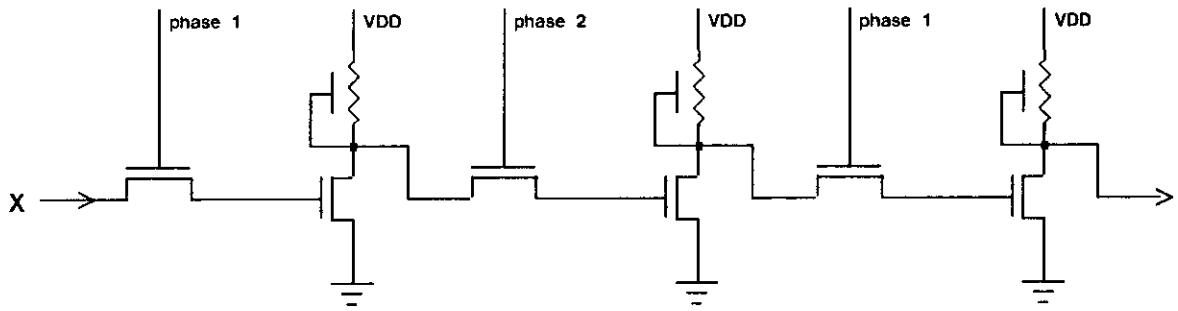


Fig. 4a Shift Register: Circuit Diagram

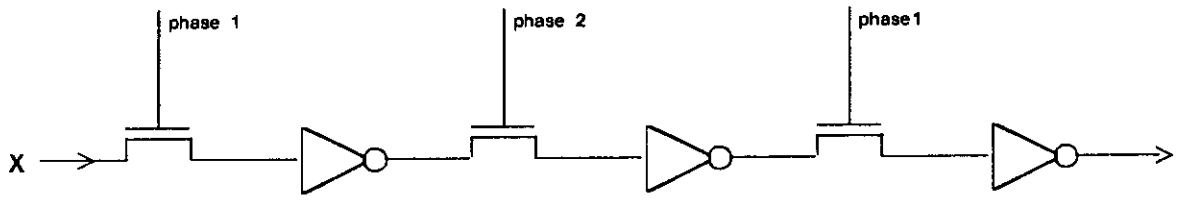


Fig. 4b. Shift Register: In Mixed Notation

The Shift Register

Perhaps the most basic structure for movement of a sequence of data bits is the *serial shift register*, shown in circuit diagram form in figure 4a. The shift register is composed of level restoring inverters coupled by pass transistors, with the movement of data controlled by applying clock signals φ_1 and φ_2 to the gates of alternate pass transistors in the sequence.

Data is shifted from left to right as follows. Suppose a logic signal X is present on the leftmost input to the shift regiter when clock signal φ_1 rises. Then, during the time when φ_1 is *high*, this signal will propagate through the pass transistor and be stored as charge on the input capacitance of the first inverter stage. For example, if the signal X is *low*, then the inverter input gate capacitance will be discharged towards zero volts during the time when φ_1 is *high*. On the other hand, if X is *high*, the inverter input capacitance will be charged up towards VDD - V_{th} during φ_1.

When the clock signal φ_1 falls, the pass transistor becomes an open circuit, isolating the charge on the input of the inverter. The second clock phase is now initiated by the rise of φ_2. During the time interval when φ_2 is *high* the logic signal X, now inverted, will flow through the second pass transistor onto the gate of the second inverter. This pattern can be repeated an arbitrary number of times to produce a shift register of any length.

Note that since the clock signals do not overlap, the successive pairs of stages of the shift register are effectively isolated from one another during the transfer of data between inverter pairs. For example, when φ_1 is *low*, and φ_2 is *high*, all adjacent inverters connected by the φ_2 controlled pass transistors are in the process of transferring data from the left to the right members of the pairs. All these pairs of inverters are isolated from each other by the intervening φ_1 controlled pass transistors which are all open circuits when φ_1 is *low*.

It is also important to note that the shortest period, T, we can use for clocks controlling such data transfers is determined by the time required to adequately charge or discharge the inverter input gate capacitance through the pass transistor and the preceding stage pullup or pulldown. To this time must then be added an increment of time sufficient to insure that the clocks do not overlap. For more complex systems, the minimum clock period may be estimated as a function of basic circuit parameters as discussed in Chapter 1.

Figures 4b and 4c illustrate the serial shift register using mixed notations. In figure 4b, each

inverter circuit diagram has been replaced by its logic symbol. In figure 4c, the pass transistor circuit symbols have been replaced by their stick diagrams. When visualizing the inverter, as represented by its logic symbol, in a circuit structure containing mainly stick diagrams, two points should be kept in mind:

(i) The input to the inverter leads directly to the gate, and thus the gate capacitance, of the inverter's pulldown transistor. This input may be used to store a data bit by isolating the charge representing the bit with a pass transistor. Note that the input path will end up on the poly level within the inverter. A contact cut may thus be required to connect the poly gate and the metal or diffusion path on which the signal enters the inverter.

(ii) Since the connection between the source and gate of the inverter pullup transistor requires a connection of all three conducting levels, the inverter output signal may easily be routed out on any one of the three levels.

Identical serial shift registers can be stacked next to each other and used to move a sequence of data *words*, as shown in figure 5a. The simple structure in figure 5a anticipates the elegant topological simplicity of many important MOS integrated system functions. By connecting the successive inverter stages with diffusion paths, the pass transistors controlled by the clock signals are formed by simply running vertical clock lines in poly. The structure in figure 5a also anticipates another important point: topological simplification often results when control signals flow on lines that are at right angles to the direction of data flow. In this way as many bits as necessary can be processed in parallel with the same control signals.

The example in figure 5a is so rudimentary it is perhaps difficult to visualize the two clock signals as actually containing control information. Let us consider a slightly more complex example, the *shift-up register array* shown in figure 5b. In this structure, each data bit moving from left to right during φ_2 has two alternative pass transistor paths through which it can proceed to the next stage: a straight through path, and a path which shifts it up to the next higher row. If the control signal SH is *low*, then $[\varphi_2 \cdot SH']$ is *high*, and the straight through pass transistor paths are used during φ_2. At the same time, $[\varphi_2 \cdot SH]$ is *low*, thus preventing data flow through the shift-up pass transistor paths. On the other hand, if SH is *high*, the straight through pass transistors are off and the shift-up pass transistor paths are used during φ_2, resulting in the entire data word being shifted vertically as well as horizontally. Here the vertical control lines are run in metal, and the pass transistors are selectively formed by crossing the appropriate diffusion paths with short poly lines.

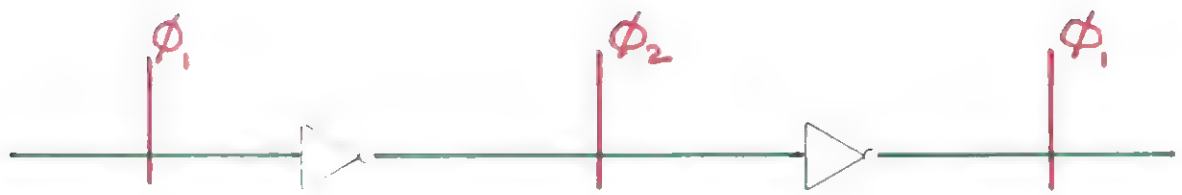


Fig.4c. Shift Register: More Mixed Notation

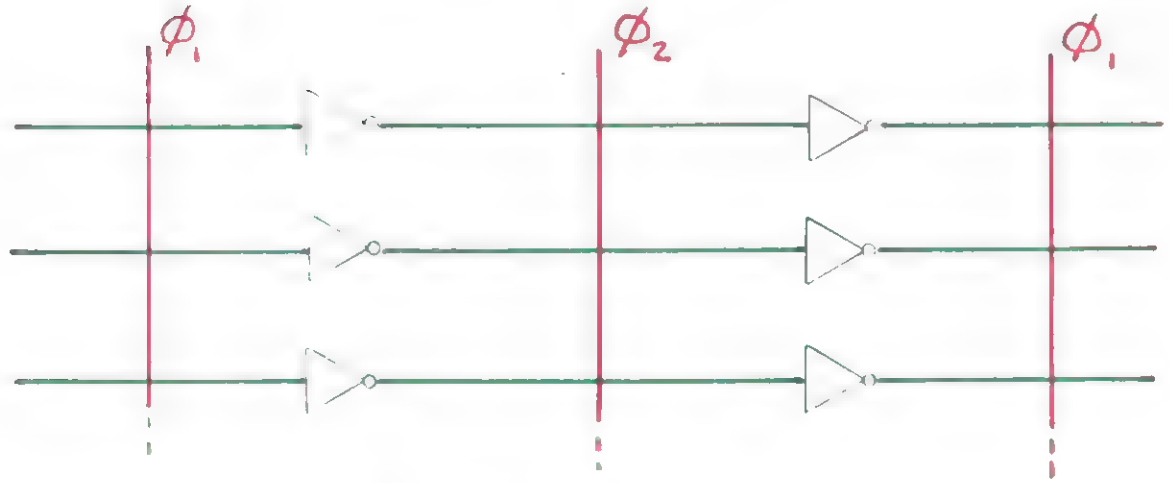


Fig.5a. Array of Shift Registers

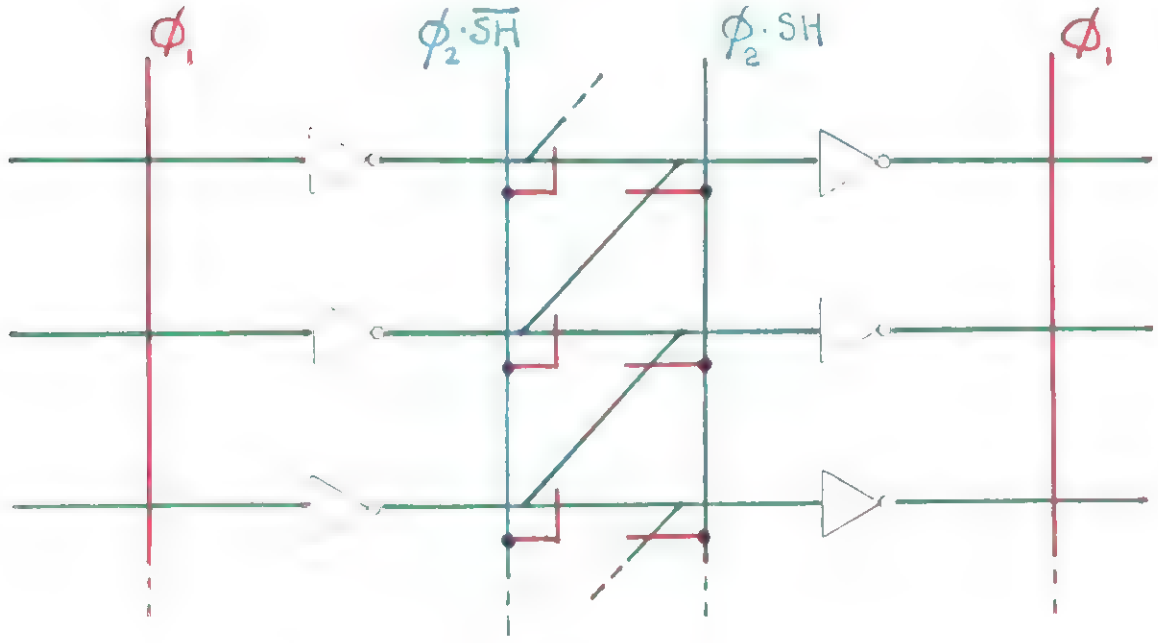


Fig.5b. Shift-Up Register Array

Relating Different Levels of Abstraction

In the discussions in this chapter, we will not have to make extensive calculations of the detailed electrical behavior of the devices and circuits involved in order to analyze the general behavior of digital logic constructed with these devices and circuits. Most of the examples presented in this chapter, and throughout the text, build upon the use of pass transistors coupling inverting logic stages as a means of structuring designs. The general results of chapter one provide the solutions to most device and circuit problems encountered, such as ratio and delay calculations, etc. In most cases, design concepts can be worked out using stick diagrams, and only at the stage of transforming the circuit topology into the detailed circuit layout geometry will these calculations need to be worked out, either by hand or with circuit simulation programs.

It is important to simplify our mental model of integrated circuitry, so as to more quickly and easily analyze or explain to others the function of a given circuit, and more easily visualize and invent new circuit structures without drifting too far away from physically realizable and workable solutions. Of course, it is in general a dangerous practice to oversimplify our abstractions of electronic circuit behavior, and there are some nMOS circuits of deceptively simple appearance which have exceedingly complex behavior. However, throughout large portions of digital integrated systems, if the circuit and subsystem design is carefully structured as suggested in this text, an extremely simple mental model of device and circuit behavior will prove adequate to predict circuit and subsystem behavior.

Figure 6a illustrates a simple way of visualizing the operation of successive inverting logic stages coupled by pass transistors. Assume for the moment that any pass transistors in the paths between stages are *on*. To visualize the time behavior of an inverter, and the effect of the pullup L/W to pulldown L/W ratio, imagine the flow of current from VDD to GND as the flow of a fluid, and the inverter's two transistors as valves. The pullup transistor is always *on*, and so the upper "valve" is always open. However, the "valve" corresponding to the pulldown transistor may be either open or closed, depending on the amount of charge on its gate.

In figure 6a, the input to inverter-A is a logic-0, so the pulldown of inverter-A is *off*, and the lower valve is closed. Current is thus diverted to the large charge storage site corresponding to the gate of the pulldown of inverter-B. If sufficient charge has flowed

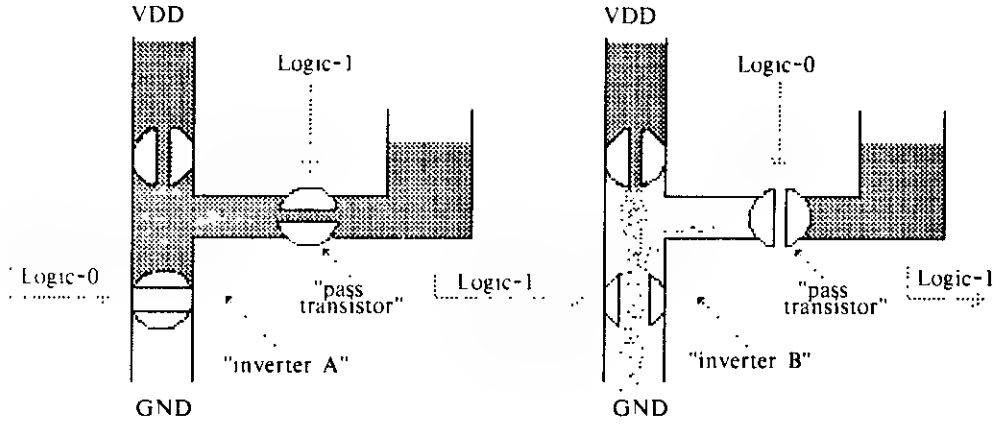


Fig. 6a. A Way of Visualizing the Operation of Successive Inverter Stages

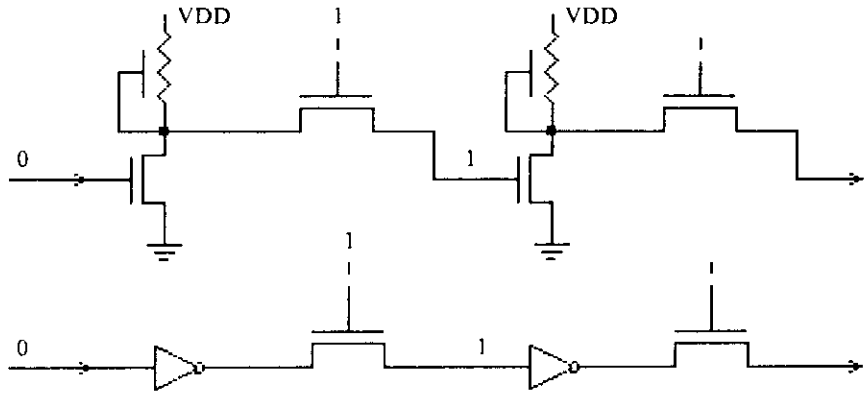


Fig. 6b. Successive Inverter Stages: Circuit Diagram and Logic Diagram

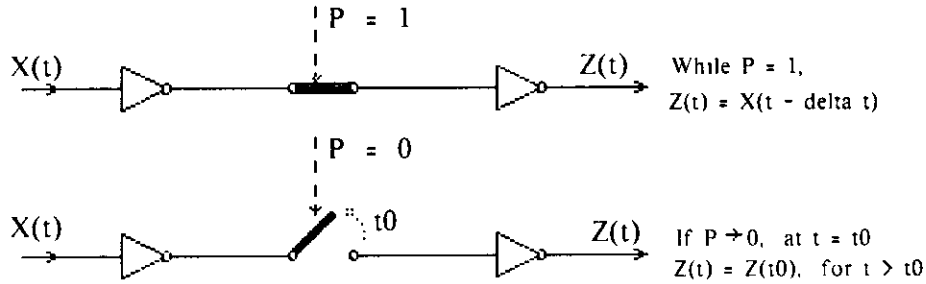


Fig. 6c. Successive Inverter Stages Connected Through a Pass Transistor

[Illustrating effect of the pass transistor "switch"]

onto this gate, corresponding to a high level of fluid in the tank representing the gate capacitance, then the pulldown of inverter-B is turned *on*, and thus the lower valve of inverter-B is open. If the lower valve in inverter-B is much larger than the upper one, corresponding to a practical pullup to pulldown size ratio, then the pulldown of inverter-B can sink all the source current provided by the pullup. In addition, if given sufficient time and if the connecting pass transistor is on, the pulldown can drain off any charge stored on the succeeding inverter's input gate. Thus we can visualize the sequence of inversions of a logic signal propagating through successive inverter stages as an alternation between high and low levels of fluid in the storage tanks. In addition, we can visualize some of the time behavior of the signal's propagation: the larger the gate capacitances (storage tanks), the longer it takes to build up enough charge to open the next stage, and similarly the longer it takes to drain charge off the next stage and thus turn it off.

Figure 6b represents the same physical circuit modelled in figure 6a, but on successively higher levels of abstraction. When analyzing circuit or logic diagrams showing successive inverting logic stages, as in figure 6b, one should keep the model of figure 6a in mind. Whether one is a novice or an expert in integrated system design, it is very helpful to compress the details of any given lower level of abstraction, so as to reduce the complexity of the problems presented at the next higher level, and enable the mind to span problems of larger scope.

We are now able to visualize a very simple model for the pass transistor: it is in fact like a valve, or "switch" in the path between an inverter and the next charge storage site, i.e. the input gate of the next inverter. Figure 6c shows two inverters coupled by a pass transistor, with the pass transistor informally symbolized as a "switch". In the upper diagram of figure 6c, the pass transistor input is a logic-1, and so the "switch" is in the *on* position, resulting in the output Z being equal to the input X, after a suitable delay time Δt. Thus during the time the pass transistor gate input P = 1, the output $Z(t) = X(t - \Delta t)$. Here Δt is some multiple of the transit time, τ, of inverter pulldown transistor, as discussed in Chapter 1.

In the lower diagram of figure 6c, the pass transistor "switch" is moved to the *off* position since P is a logic-0. Therefore, according to our model, the valve in the path is shut, and the charge, or lack of charge, is isolated in the storage site. Thus once the pass transistor valve is shut, Z remains at a constant value, independent of changes in X: i.e., if $P \rightarrow 0$, at $t = t_0$, then $Z(t) = Z(t_0)$, for $t > t_0$.

These simple visualizations of the inverter and the pass transistor will carry us fairly far into LSI subsystem design. Several logic circuits in this chapter are drawn first in stick diagram form, and then informally sketched with pass transistors replaced with "switches", both to clarify the behavior of the circuits involved, and to further demonstrate the applicability of the model.

Implementing Dynamic Registers

Registers for the storage of data play a key role in digital system design. It is interesting to note that a group of adjacent inverters, with their gates isolatable by pass transistors, can be considered a form of temporary storage register. This arrangement is illustrated in figure 7, which shows two levels of symbolism for this *dynamic register*. Such a register is very simple in structure. It consists of only three transistors per bit position: the pass transistor and the two transistors of the inverter. However, this dynamic form of register will preserve data only as long as charge can be retained on the inverter input gates. Typically dynamic registers are used in situations where the input gate updating control signals are applied frequently. They are ideal in a clocked system in which they are reloaded every clock cycle, as in the shift register.

Suppose we wish to construct a simple register which can be loaded during the appropriate clock phase under the control of a *load* signal, and which will retain its information through an indefinite number of successive clock periods until it is reloaded using the *load* signal. A one bit cell for such a register may be constructed using cross coupled inverters in the configuration shown in figure 8. This register cell is still dynamic in form, since it uses charge storage on the gate of the first inverter to preserve its state. However, it need not be loaded on every successive φ_1 as was the simple register in figure 7. The pass transistor leading to it from the preceding stage is switched on only when *both* φ_1 and LD are *high*. On any following φ_1 when LD is *low*, the cell updates itself by the feedback path through the second pass transistor. Figure 9 illustrates a selectively loadable register composed of such cells. One important feature of this type of register is that it provides as output both the true and complemented forms of the stored data. This feature is often useful when the data is to be processed by a following network of combinational logic.

While there are more elaborate forms of dynamic and static registers, the above two forms are sufficient for many of the required data storage applications within integrated systems.

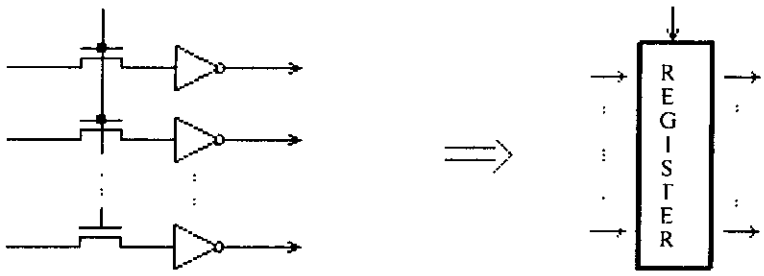


Fig. 7. A Dynamic Register

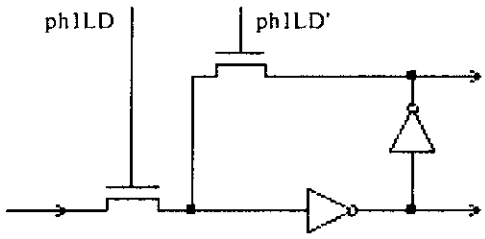


Fig. 8. A Selectively Loadable Dynamic Register Cell

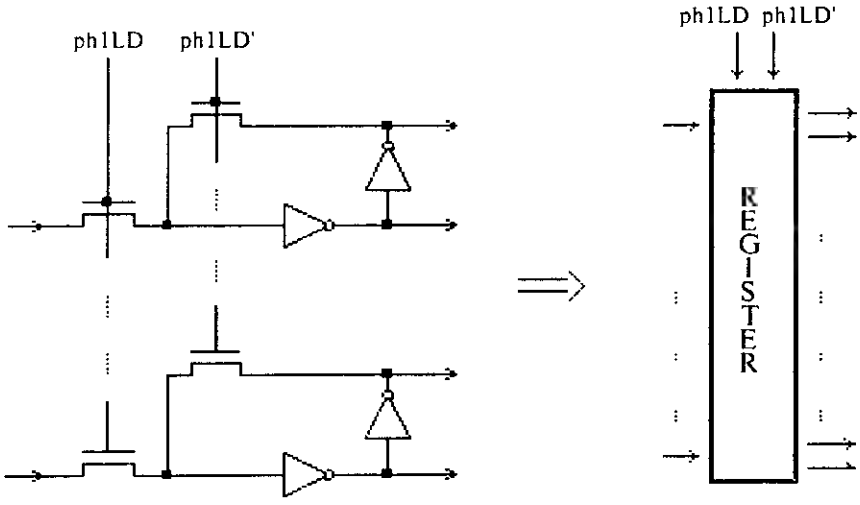


Fig. 9. A Selectively Loadable Dynamic Register

Implementing a Stack

The ideas used to construct simple dynamic registers in the preceding section may be applied to the construction of more sophisticated and interesting subsystems. In this section we will describe the implementation of a *stack*. This type of subsystem is commonly called a *last-in, first-out* stack (LIFO), or *pushdown* stack, although we will diagram it horizontally rather than vertically. It is a shift register array with three basic operations: during each full clock period (1) we can *push* in a new data word at one end of the array, pushing all previously entered words one word position further into the array, or (2) we can leave all words in their current position, or (3) we can *pop* out a word from the end of the array, pulling all previously entered words back out by one word position.

Figure 10a shows the structure of one horizontal row of the stack. Here we have implemented a shift register which can perform the following three operations: shift data left to right, hold data in place, or shift data right to left. There are four control signals used, two of them being active during φ_1 and two of them being active during φ_2. The signals φ_1 and φ_2 are our familiar two phase, non-overlapping clock signals.

In order for data to be shifted from left to right, the shift right control line (SHR) is driven *high* during φ_1, followed by driving the transfer right control line (TRR) *high* during φ_2. The bit of data appearing at the left is thus transferred by this operation onto the gate of the first inverter during φ_1, and thence to the gate of the second inverter during φ_2. In order for data to be held in place, the signal transfer left (TRL) is driven *high* during φ_1 and transfer right (TRR) is driven *high* during φ_2, causing the data to recirculate upon itself without shifting. Note that the data can be obtained at any time from the output of the first inverter. However, since new data may come to the gate of the first inverter during φ_1, the only safe time to take data out to the left is during φ_2. The transfer of data from right to left is caused by driving the shift left control (SHL) line *high* during φ_2, followed by driving transfer left (TRL) *high* during φ_1.

Figure 10b illustrates a possible topological structure of one horizontal row of the stack. There are two horizontal pathways on the diffusion level for shifting bits right or left. The two inverters for one stage of the row are nested between these paths. VDD, GND, and the four control lines run vertically in metal. The four pass transistors required for controlling the movement of data are conveniently implemented by short poly lines which cross the horizontal diffusion tracks at appropriate positions. Note that the entire row is composed

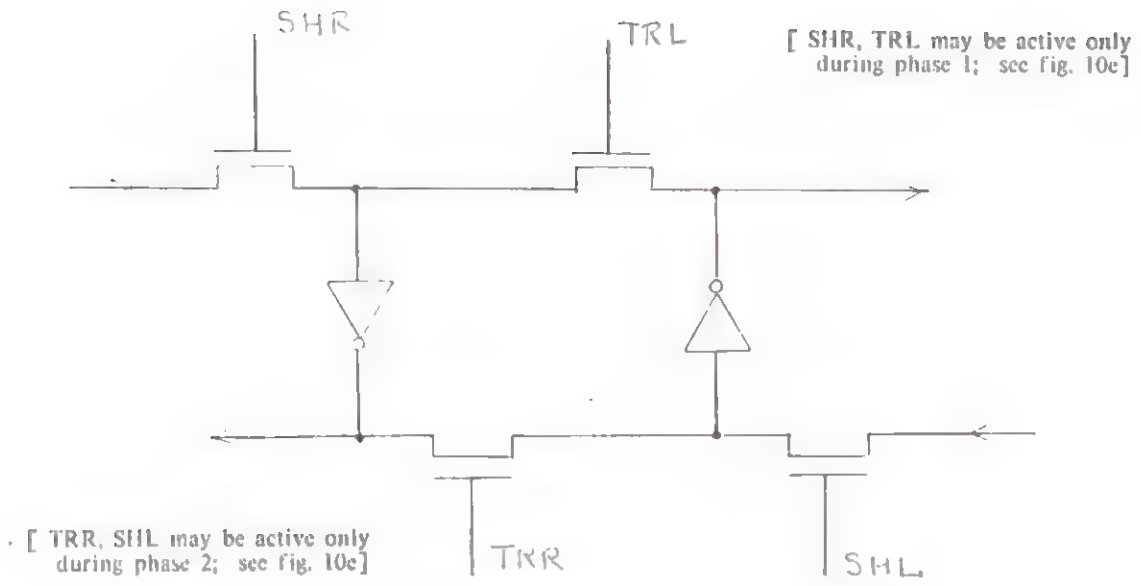


Fig.10a. One Horizontal Row of the Stack

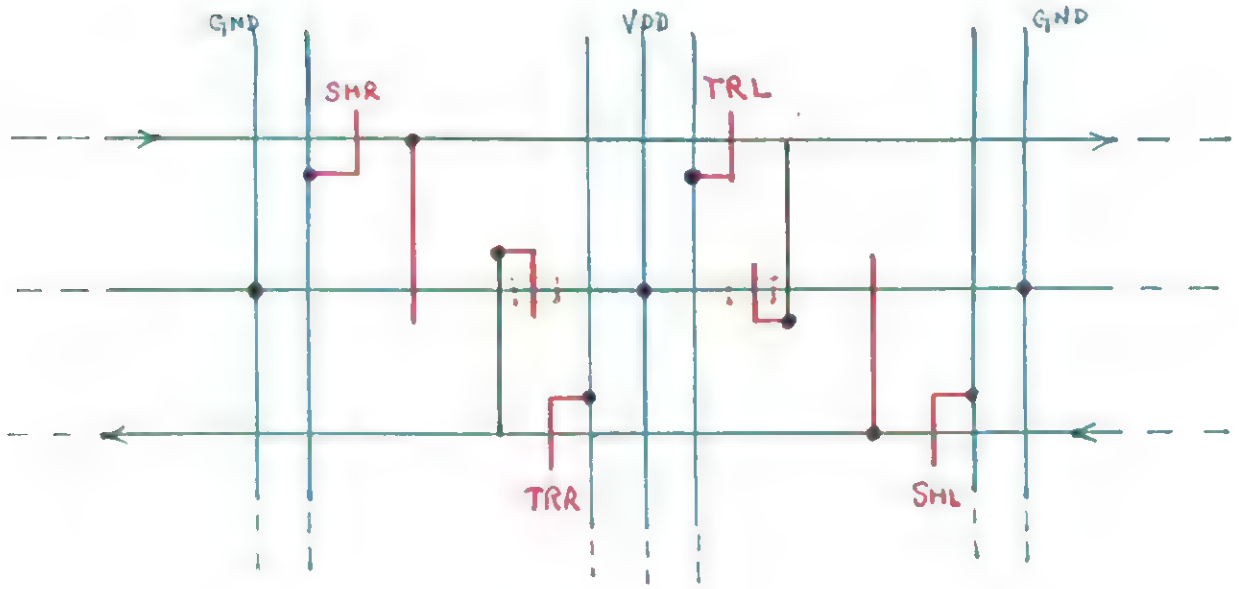


Fig.10b. Topology of One Horizontal Stack Row

of 180° rotations and repetitions of a basic cell containing one inverter.

In a typical implementation of the complete LIFO stack, a number of such rows run parallel to each other in the horizontal direction. The number of rows is equal to the width in bits of the data words involved. The control lines run vertically across the entire stack, perpendicular to the direction of data flow. For data words of any substantial width, the capacitive loading on the control signals would be sufficient to warrant use of super-buffer drivers.

The stack as a whole may be controlled with only two logic signals: one signalling *push*, and the other signalling *pop*. The activation of neither of these two signals causes data to recirculate in place, awaiting the next active instruction.

Let us consider how to derive, from *push* and *pop*, the control signals for driving the four control lines SHR, TRR, SHL, TRL. A possible scheme is shown in Fig. 10c. We use random logic for this purpose since only a few gates are required to control the large, regular array of circuit cells in the stack. The operation which determines what the stack will do during the subsequent clock phase is brought in on the path labeled OP. It is important to note in the following that only one signal path (OP) is required to bring in both *push* and *pop* logic signals, since these are active on mutually exclusive clock phases.

The control scheme is summarized in the timing diagrams in figure 10e. Here we see that holding OP *high* during φ_2, followed by *low* during φ_1, implements *push*. Holding OP *low* during both φ_1 and φ_2 causes the data to recirculate in place. Holding OP *high* during φ_1, followed by *low* during φ_2, implements *pop*. Thus, the single signal path, OP, is sufficient to carry both stack control signals into the stack.

During φ_1, the OP signal is fed through the upper pass transistor into the inputs of the two NOR gates g_1 and g_2. The outputs for these two NOR gates are *low* during this period, since φ_1 is *high*.

If the incoming OP signal is *high* while φ_1 is *high*, then the lower input of NOR gate g_2 will be *low*. Thus when φ_1 falls *low*, the output of g_2 will go *high*, thereby driving SHL *high*. If the OP signal is instead kept *low* while φ_1 is *high*, then the output of the NOR gate g_1 will go *high* on the fall of φ_1, thereby driving TRR *high*.

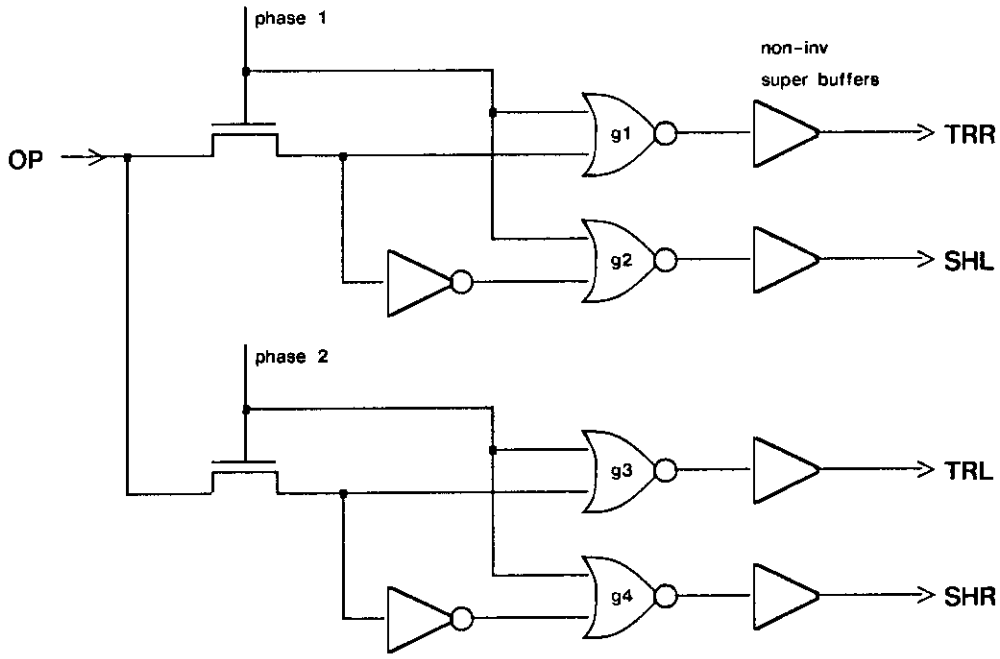


Fig. 10c. Generating the Stack Control Signals

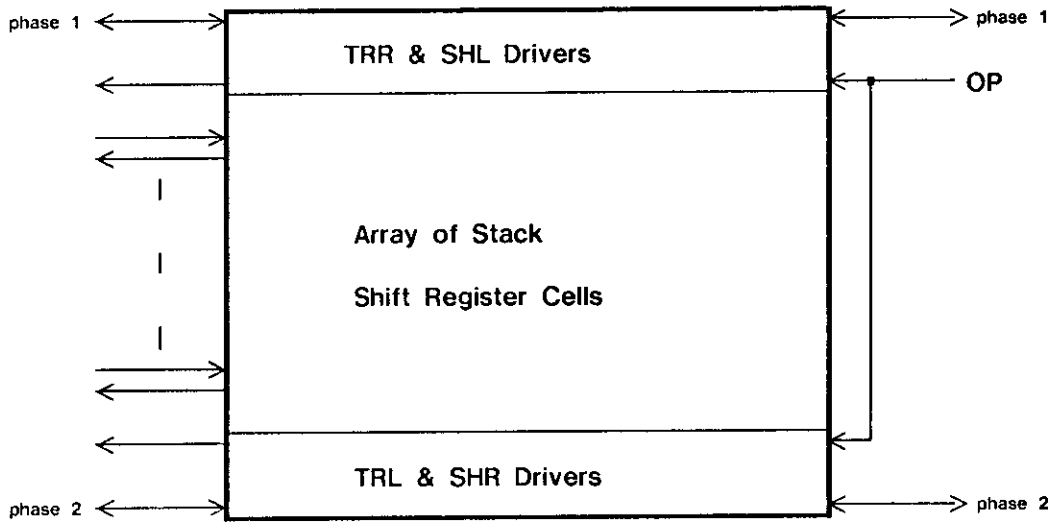


Fig. 10d. Stack Geometry and Interconnect Topology

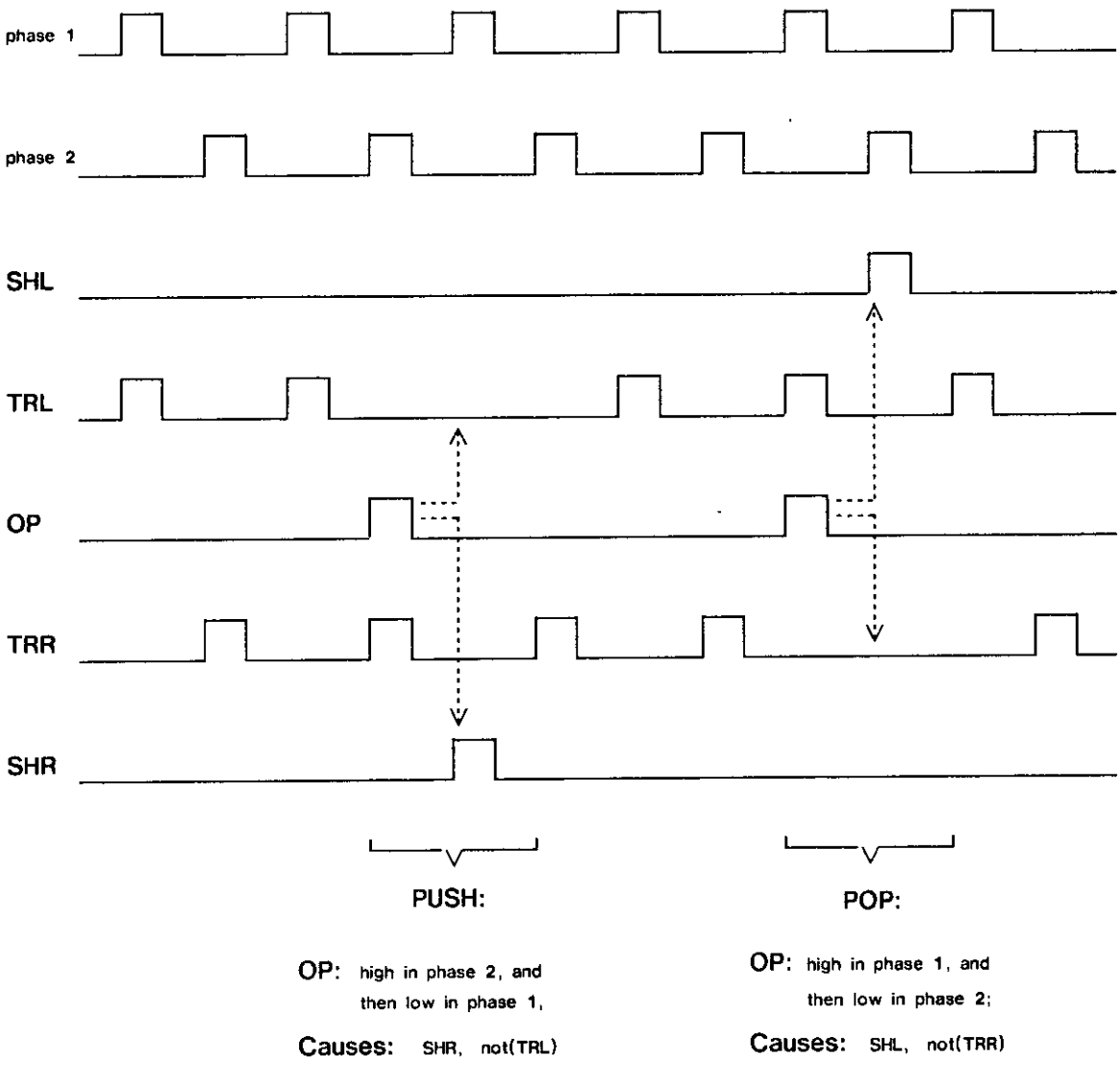


Fig. 10e. Stack control Signal Timing Diagrams

(dcf10e sil)

During the period when φ_2 is *high* and either the shift left (SHL) or the transfer right (TRR) operation is being executed, the signal on the OP line is being stored on the corresponding input gates of the lower two NOR gates, g_3 and g_4. Thus, if OP is *high* while φ_2 is *high*, a logic-0 is stored on the input of the NOR gate g_4, and during the subsequent φ_1 *high* period, SHR will be driven *high*. Conversely, if OP is *low* while φ_2 is *high*, TRL will be driven *high* during the following φ_1 *high* period.

This kind of control scheme recognizes that there must be a lull period between any operation and its next occurrence. Control information is taken in during this period and set up for the subsequent operation. The scheme takes advantage of these lull periods, when possible, to perform other operations which can be done without conflict. It is an example of a fundamental design technique which can be extended to larger system structures.

When planning the overall architecture of a larger system, it is often useful to represent subsystems, such as the stack, using a higher level of symbolism. To be truly useful, such representations should, in addition to a functional definition, include the *topological* factors associated with the interconnection points of the subsystem and the *geometrical* factors of its shape and relative physical dimensions.

A system level sketch of one particular implementation of the stack is shown in figure 10d. Identical driver circuitry is placed along the top and bottom edges of the shift register array. The transfer right and shift left drivers which are set up during φ_1 (and active during φ_2) are placed along the top of the shift register array. The transfer left and shift right drivers which are set up during φ_2 (and active during φ_1) are placed along the bottom of the array. This choice has made the two timing signals local to the drivers which they control. The OP bit is required on both the top and the bottom of the shift register array. Equivalently, *push* is required only on the bottom of the array, while *pop* is required only on the top.

The integration of this subsystem into a larger integrated system design will require that the data in and out paths be matched to those of subsystems to which the array is connected, and that the φ_1, φ_2, and OP signals be available at either the left or right side of the array. By using system level representations that reflect as closely as possible the dimensions and locations of critical signals in all major subsystems, the interactions between topologies and dimensions of the subsystems can be assessed. The feasibility of an overall system architecture can thus be ensured prior to detailed design and layout.

Register to Register Transfer

From an implementation point of view it is often desirable to combine logic steering functions with the clocking of data into registers, since both require pass transistors as their elementary functional unit. An example is the shift-up register array shown in figure 6. From the next higher level system view, however, it is desirable to separate the two functions conceptually. In Fig. 11a we have shown some combination of inputs, X0 through Xn going through some combination of pass transistors, *which may or may not have logic functions attached*, into the input gates of some inverting logic elements. This combination of function is then abstracted into a register clocked on the phase during which the input pass transistors are turned on. Any logic function associated with the input pass transistors is considered part of the preceding combinational logic module. This viewpoint is an extension of the concept of dynamic register previously developed in figure 7.

Using this notation, any processing function can be built up using blocks of the form shown in Fig. 11b. Here we have a clocked input register, a block of strictly combinational logic *with no timing attached*, and an output register clocked on the opposite phase. In this case the inputs are stored in the input register during φ_1. They then propagate into and through the combinational logic (C/L), with the resulting outputs stored in the output register during φ_2. Any single data processing step can be viewed as a transfer from one such register to a second through a combinational logic block.

A sequence of such operations can be performed on a data stream by a series of such combinational blocks separated by registers as shown in Fig. 11c. Since different sets of data words in the stream may be operated upon at the same time, but at different locations, this data path is a type of pipelined processing structure. Such pipelined processing structures offer the opportunity for improved processing bandwidth by performing many different operations concurrently. Notice that the throughput rate of such a pipeline system of register to register transfer operations is limited by the delay time through the slowest of the combinational logic blocks. If no registers had been interposed between the function blocks, and each operand set separately run through the entire sequence of combinational logic modules, the throughput rate would be much lower.

In line with the ideas developed earlier in this chapter, the detailed functions performed by the combinational logic modules may often be implemented in circuit structures of very simple and regular topology. Control signals will in general cross the data path at right

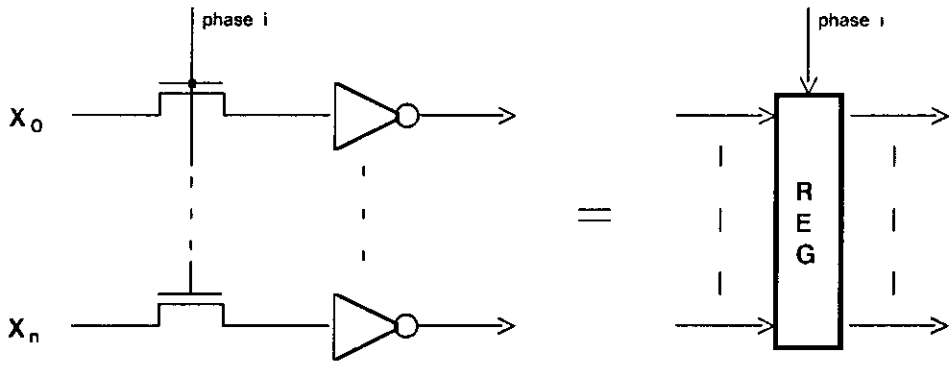


Fig. 11a. A Register

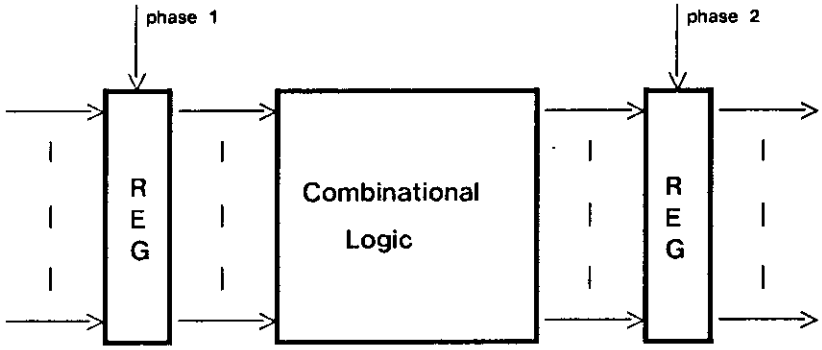


Fig. 11b. A Section of a Data Path

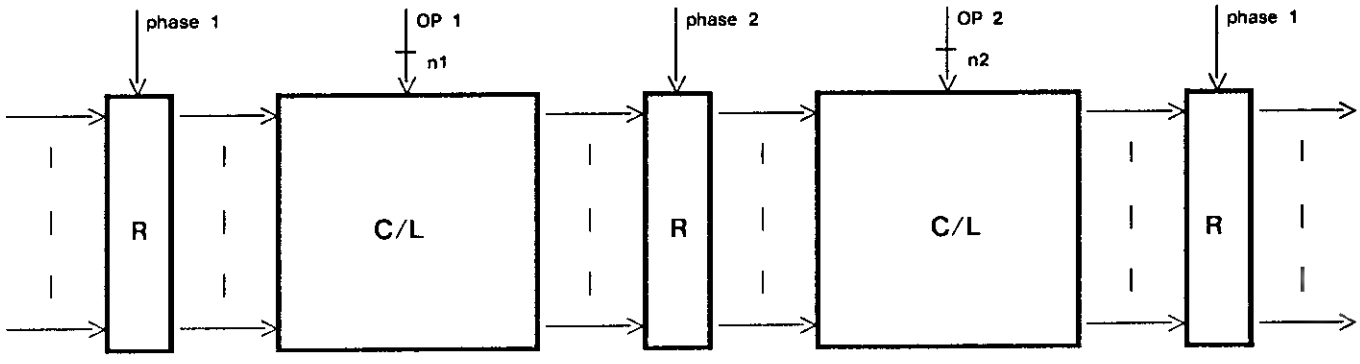


Fig. 11c. General Form for a Data Path

angles to the direction of data flow. Figure 11c illustrates sets of such control inputs as n_1 lines carrying the control function OP_1 into the first C/L module, n_2 lines carrying OP_2 into the second, etc.

The idea of data being processed while passing through combinational logic interspersed between register stages in a sequence of register to register transfers is a basic and important concept in the hierarchy of digital system architecture. We have already described the implementation of registers. The next sections will describe some ways to implement combinational logic functions.

Combinational Logic

Combinational logic modules contain no data storage elements. The outputs of a combinational logic module are functions only of the inputs to that module, provided that sufficient time has been allowed for those inputs to propagate through the module's circuitry.

In integrated systems, combinational logic design problems will typically fall within one of three general classes. The first is when a small amount of simple logic is required, for example to derive control signals at the periphery of a system module (as in the stack control signal generation) or to implement a simple function within a single circuit cell (which may then be replicated in a regular array). In these cases, traditional logic design procedures using static NAND and NOR gates can be applied. Such designs involving a few gates are usually rather simple, and can be produced by inspection rather than by use of formal minimization and synthesis procedures. Even in these simple cases, the minimum static logic gate implementation does not necessarily result in either the most regular, the minimum area, the minimum delay, or the minimum power design. In fact, we often find alternative techniques to the use of static logic gates, which in specific instances lead to "better" designs by one of these measures than would minimum gate implementations. For example, figure 12a shows a *selector* logic circuit (I. Sutherland), in which one of the inputs S_1, S_2, S_3, S_4 is selected for output by the control variables A, and B according to the function:

$$f = S_1A'B' + S_2AB' + S_3A'B + S_4AB$$

This selector circuit is composed simply of poly paths crossing diffusion paths. Where depletion mode transistors are placed, the diffusion level path is always connected, thus placing control in the selectively located enhancement mode pass transistors, which function as simple switches. Figure 12c shows the circuit's paths from inputs to outputs using the "switch" abstraction for each of the pass transistors. For each possible combination of values of A and B, there is a path through the selector to Z from only one of the inputs S_i. For the specific inputs shown in the example in figure 12c, the signal S_2 propagates through to *Z* since both A and B' are *high*. Note that no static power is consumed by the circuit, and the area occupied by the circuit is minimal since no contact cuts are required within it. In chapter 5 we describe a very general and powerful arithmetic logic unit (ALU) which uses an array of such selector blocks to control a pass transistor carry network.

The second general class of combinational logic design problems are those rather complex functions for which clever ways of structuring topologically regular implementations have been discovered. As an example, consider the implementation of a *tally* function. This function has n inputs and n+1 outputs. The k^{th} output is to be *high*, and all other outputs *low*, if k of the inputs are *high*. The boolean equations representing this function for the simple case of three inputs are:

$$Z_0 = X_1'X_2'X_3'$$

$$Z_1 = X_1X_2'X_3' + X_1'X_2X_3' + X_1'X_2'X_3$$

$$Z_2 = X_1X_2X_3' + X_1X_2'X_3 + X_1'X_2X_3$$

$$Z_3 = X_1X_2X_3$$

If this function were designed with random logic consisting of active pullup, static logic gates, it would result in a topological kludge. Figure 12b shows a topologically regular implementation of the tally function. A major portion of the function is implemented using a regular array of identical cells each containing only two pass transistors. The design is based on the shift-up register idea presented earlier. A *high* signal propagates through the array from the pullup at the lower left. Whenever one of the variables X_i is *high*, the propagating *high* signal moves up to the next higher horizontal diffusion level path. Thus the number of paths it moves up equals the number of inputs X_i which are *high*. Logic-0 signals propagate through the array from the ground points to all other outputs.

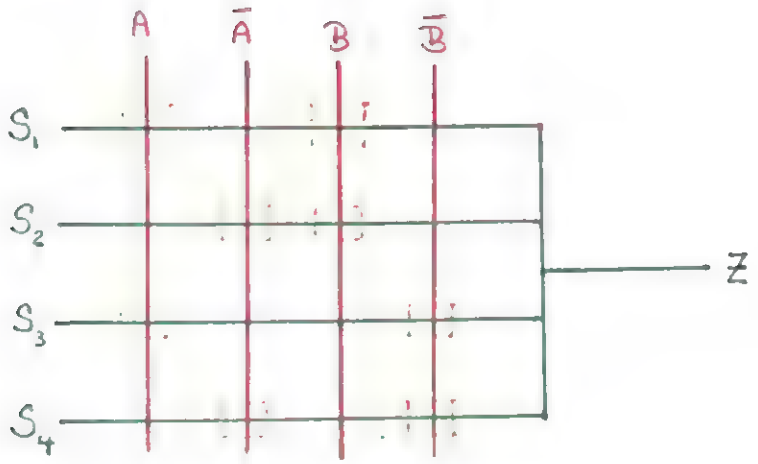


Fig.12a. Selector Logic Circuit

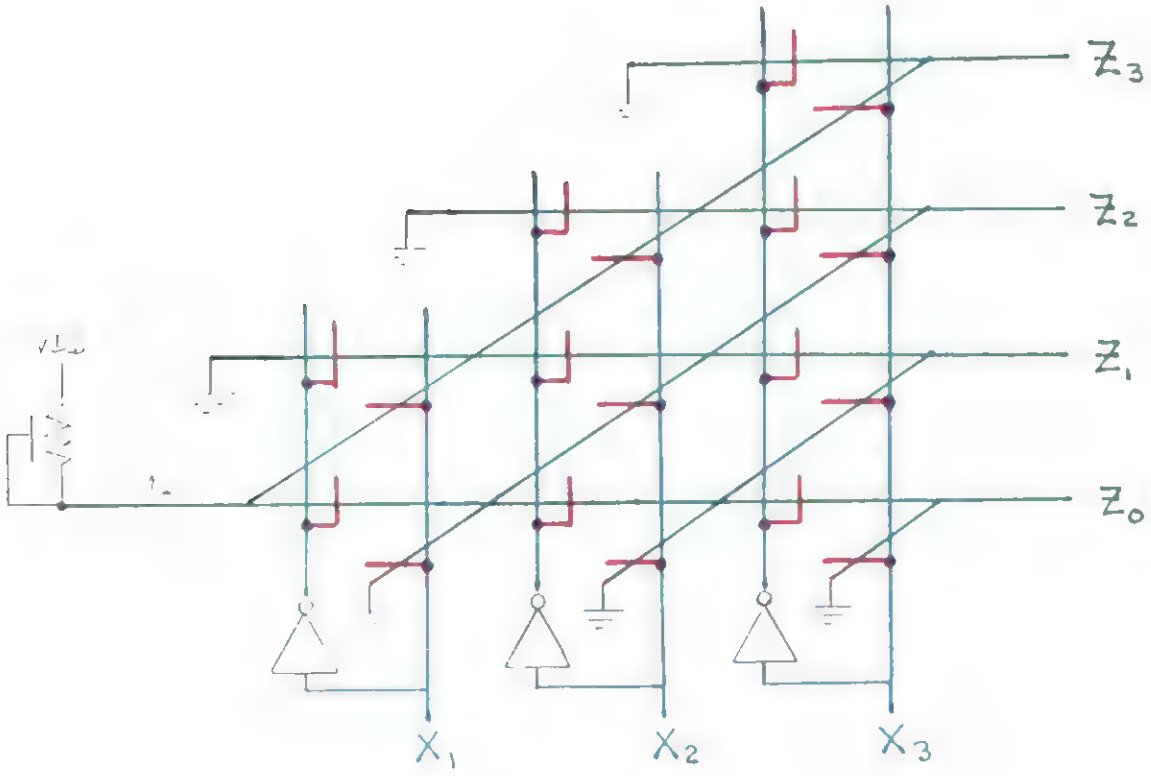


Fig.12b. A Tally Circuit

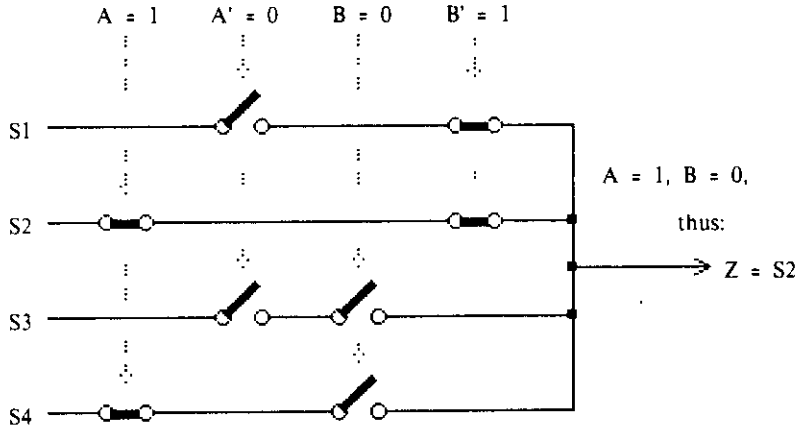


Fig. 12c. Example of Operation of Selector Circuit

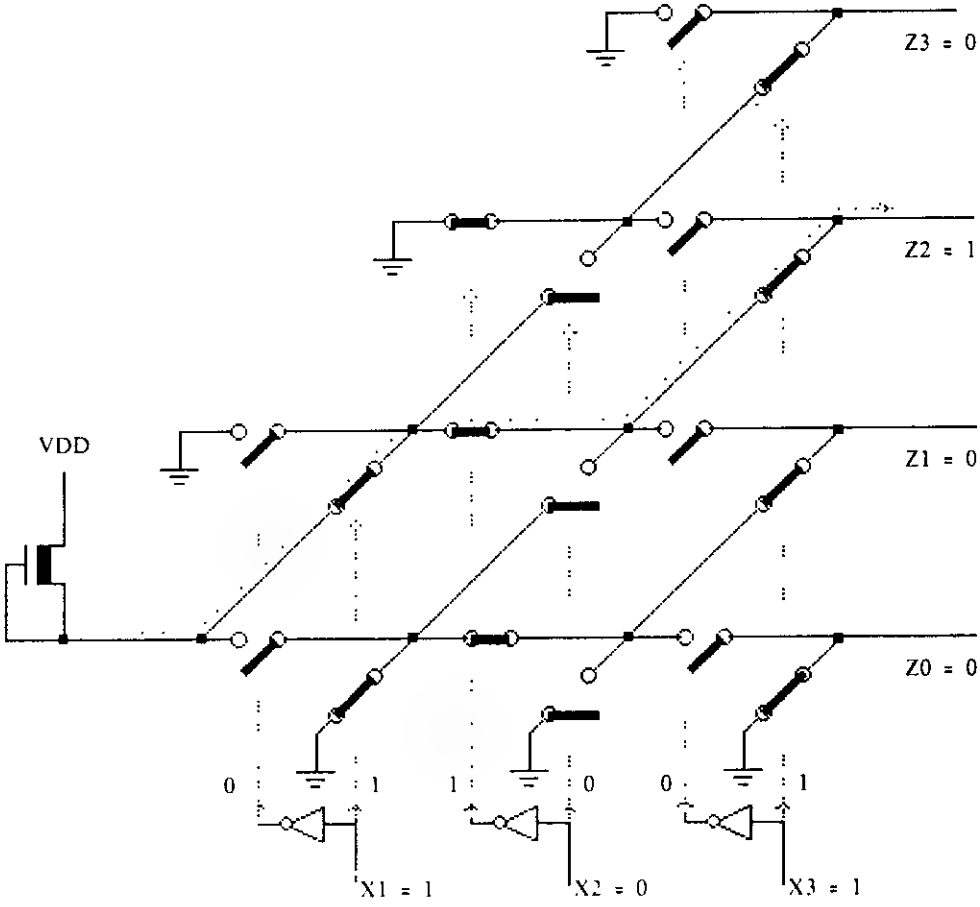


Fig. 12d. Example of Operation of Tally Circuit
(visualizing where the switches are)

(dcf12cd.press)

Figure 12d shows the paths from inputs to outputs for this tally circuit, using the "switch" abstraction for the pass transistors. The figure shows a specific example of a set of inputs controlling the pass transistors of the circuit. Since two of the inputs are *high*, the logic-1 signal is shifted up two rows and emerges at Z_2.

This tally function design can be easily expanded to handle more than three inputs by simply extending the array structure upwards and to the right. However, remember that the delay through n pass transistors is proportional to n^2. Thus it may be necessary to insert level restoration prior to such extension. Similar comments apply to the extension of the selector circuit previously shown, or other pass transistor logic arrays one might invent.

The electronic logic gates traditionally used in digital design are unilateral elements: they allow a logic signal to propagate in one direction only. It should be noted that the pass transistor is a bilateral circuit element. It permits the flow of current, and thus the passage of a logic signal, in either direction when its gate is *high*. While this property of the pass transistor is not necessarily of fundamental importance in integrated systems, it is an interesting and occasionally useful one.

Early relay switching logic used switching contacts which were bilateral elements. Interesting discussions of relay switching logic are contained in both references R4 and R5. The tally array example just given is a basic *symmetric network* mapped directly into nMOS from relay switching logic (see R5, p.241). The mathematics of switching universally used in digital systems today was proposed by Claude Shannon (R7) in 1938. Shannon demonstrated that the calculus of propositions, based on the algebra of logic developed by Boole (R8), was directly applicable to relay switching circuits.

The third and final combinational logic design situation occurs when a complex function must be implemented for which no direct mapping into a regular structure is known. Methods for handling this situation are the subject of the next section.

In the design methodology developed in this text, the combinational logic between stages in the register to register transfer paths is often done by operations on the *charge* moving between stages, using pass transistors to perform these operations. Many researchers at the present time are searching for alternative structures and techniques for performing these elementary functions, including the use of charge transfer devices[5].

The Programmable Logic Array

On many occassions it is convenient to implement the combinational logic interspersed between register stages with regular structures of pass transistors. However, we will often encounter important combinational logic functions which do not map well into such regular structures. In particular, combinational logic used in the feedback paths of finite state machines is often highly complex and inherently irregular. Also, we may wish to delay binding the details of the logic functions used in finite state machine sequencing until most of the design is complete. If the combinational logic were implemented in an irregular structure, such changes could require a major redesign.

Fortunately, there is a way to map irregular combinational functions onto regular structures, using *programmable logic arrays* (PLA's) as described in this section. This technique of implementing combinational functions has a great advantage: functions may be significantly changed without requiring any major design or layout changes of the PLA structure.

One very general and regular way to implement a combinatorial logic function of n-inputs and m-outputs is to use a memory of 2^n words of m-bits each. The n-inputs form an address into the memory, and the m-outputs are the data contained in that address. Such a memory implements the full truth table for the output functions. Many systems are in fact built using memories as combinational logic elements. A common form of memory for this purpose is the *read-only memory* (ROM) where the data is permanently placed in the memory by a mask pattern, or by electrically altering the individual bit positions. There is one major difficulty with this approach: it is often the case that most of the possible input combinations cannot occur, due to the nature of the specific problem. Stated another way, many combinational logic functions require only a small fraction of all 2^n product minterms for a canonical sum of products implementation. In such cases, a ROM is very wasteful of area.

The *programmable logic array* (PLA) is a structure which has all the generality of a memory for implementing combinational logic functions. However, any specific PLA structure need contain a row of circuit elements only for each of those product terms that are actually required to implement a given logic function (the essential prime implicants; R4, Ch.4). Since it does not contain entries for all possible minterms, it is usually far more compact than a ROM implementation of the same function. To achieve full compaction,

the various output functions must be jointly minimized before the PLA layout pattern can be defined. However, such minimization is not essential. Less than full compaction increases the independance of the different entries, so that changes in function may require only local changes in the PLA.

An illustration of the overall structure of a PLA is shown in figure 13a. The diagram includes the input and output registers, in order to show how easily these are integrated into the PLA design. The inputs, stored during φ_1 in the input register, are run vertically through a matrix of circuit elements called the AND-plane. The AND-plane generates specific logic combinations of the inputs and their complements. The outputs of the AND-plan leave at right angles to its inputs and run horizontally through another matrix called the OR-plane. The outputs of the OR-plane then run vertically and are stored in the output register during φ_2.

The circuit diagram of a specific programmable logic array is shown in figure 13b. This diagram will help to clarify the structure and function of the AND and OR-planes of the PLA. The input register bit for each input path is formed by a pass transistor clocked on φ_1 leading to both inverting and non-inverting super buffers. These buffers drive two lines running vertically through the AND-plane, one for the input term and one for its complement. The outputs of the AND-plane are formed by horizontal lines with pull-up transistors at their leftmost end. The function of the PLA's AND-plane is then determined by the locations and gate connections of pull-down transistors connecting the horizontal lines to ground.

Each output running horizontally from the AND-plane carries the NOR combination of all input signals which lead to the gates of transistors attached to it. For example, the horizontal row labelled R_3 has three transistors attached to it in the AND-plane, one controlled by A, one by B and one by C'. If any of these inputs is *high*, then R_3 will be pulled down towards ground and will be *low*.

Thus, $R_3 = (A + B + C')' = A'B'C$. Similarly, $R_4 = (A + B' + C)' = A'BC'$.

The OR-plane matrix of circuit elements is identical in format to the AND-plane matrix, but rotated 90 degrees. Once again, each of its outputs is the NOR of the signals leading to the gates of all transistors attached to it. In figure 13b for example, both R_3 and R_4 lead to the gates of transistors leading from the output line '4' to ground. If either R_3 or R_4 is

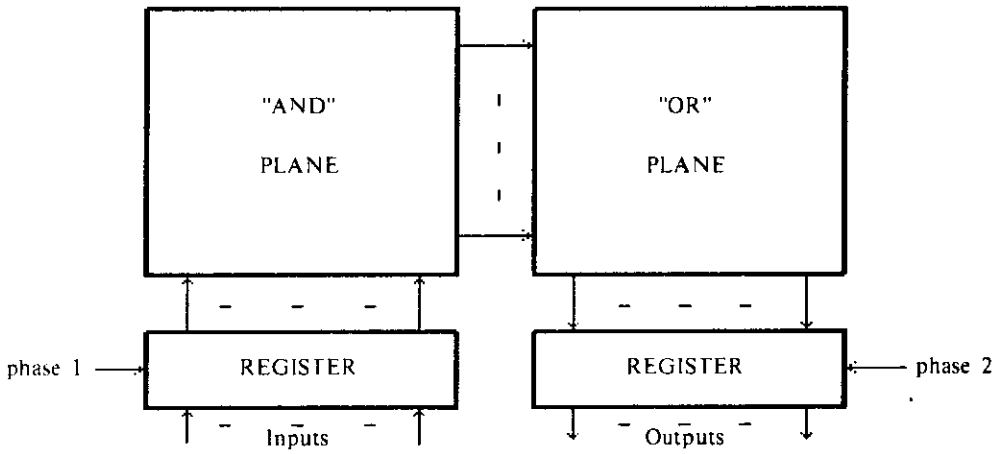


Fig. 13a. Overall Structure of the PLA

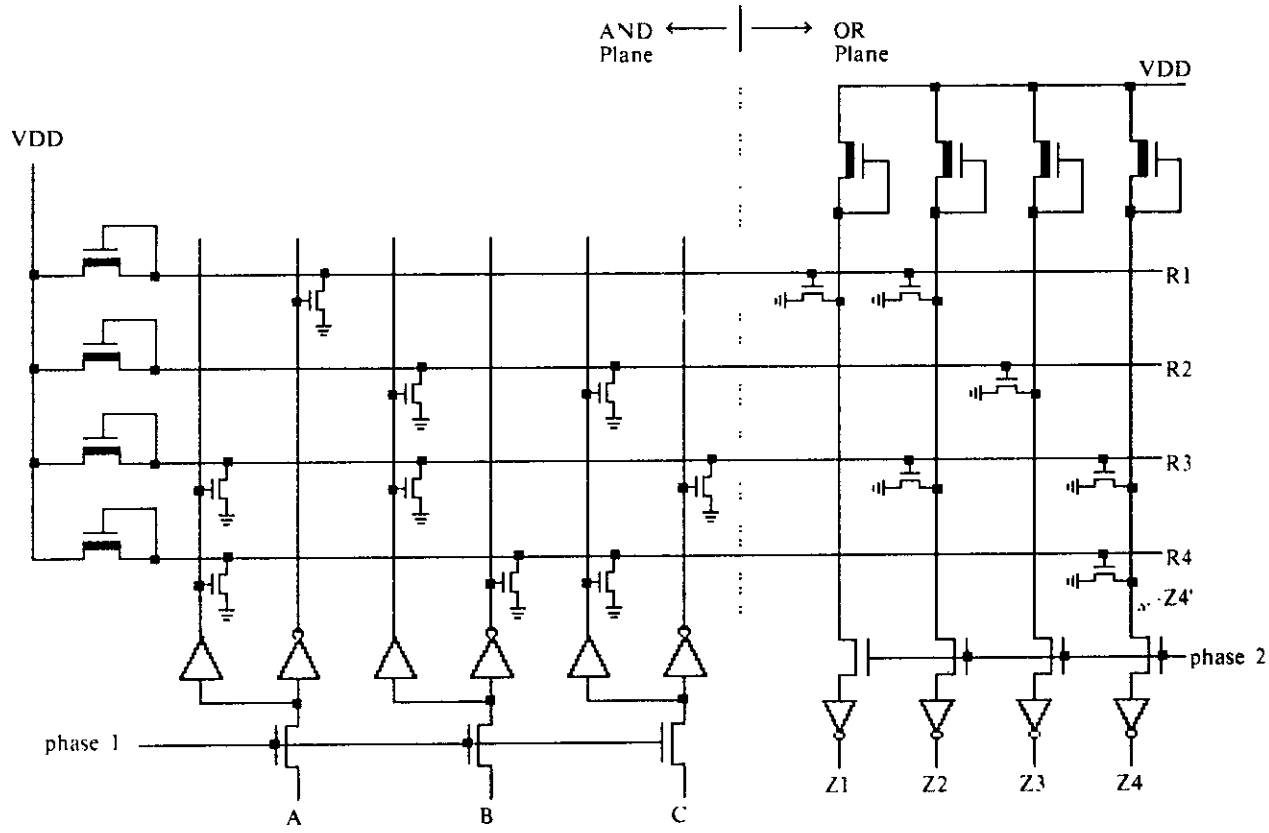


Fig. 13b. Circuit Diagram of PLA Example

high, Z_4' will be *low*. Thus, $Z_4' = NOR(R_3,R_4) = (A'B'C + A'BC')'$. Up to this point the PLA implements the *NOR-NOR canonical form* of boolean function of its inputs.

The output lines of the OR-plane matrix are run into an output register formed by pass transistors (clocked on φ_2) leading into inverting drivers. Note that the output Z_4 at this point is: $Z_4 = A'B'C + A'BC$. This expression illustrates why the two PLA planes, each implementing the NOR function, are usually referred to as the AND and OR-planes. Following the output register, the outputs appear directly as the *sum of products canonical form* of boolean functions of the PLA inputs, that is, the OR of AND terms. Each horizontal line of the PLA carries one *product term*. The number of horizontal lines is minimized when the product terms they carry are the essential prime implicants of the output functions.

Figure 13c shows one possible layout topology for implementing the PLA in nMOS circuitry. The example is the same circuit illustrated in figure 13b. The input lines crossing each plane are run in poly. The output lines from each plane are run in metal. Paths running to ground are placed between alternate poly lines, on the diffusion level. It is then a simple matter to form the pulldown transistors connecting the metal output lines to ground. They are selectively located diffusion lines under the appropriate input poly lines.

Although the PLA may implement a very irregular combinational function, the irregularity is confined to the irregular locations of pulldown transistors which "program" the function. The overall structure and topology of the PLA are very regular. Note that its overall shape and size is a function of the parameters: (i) the number of inputs, (ii) the number of product terms, (iii) the number of outputs, and (iv) the length unit λ.

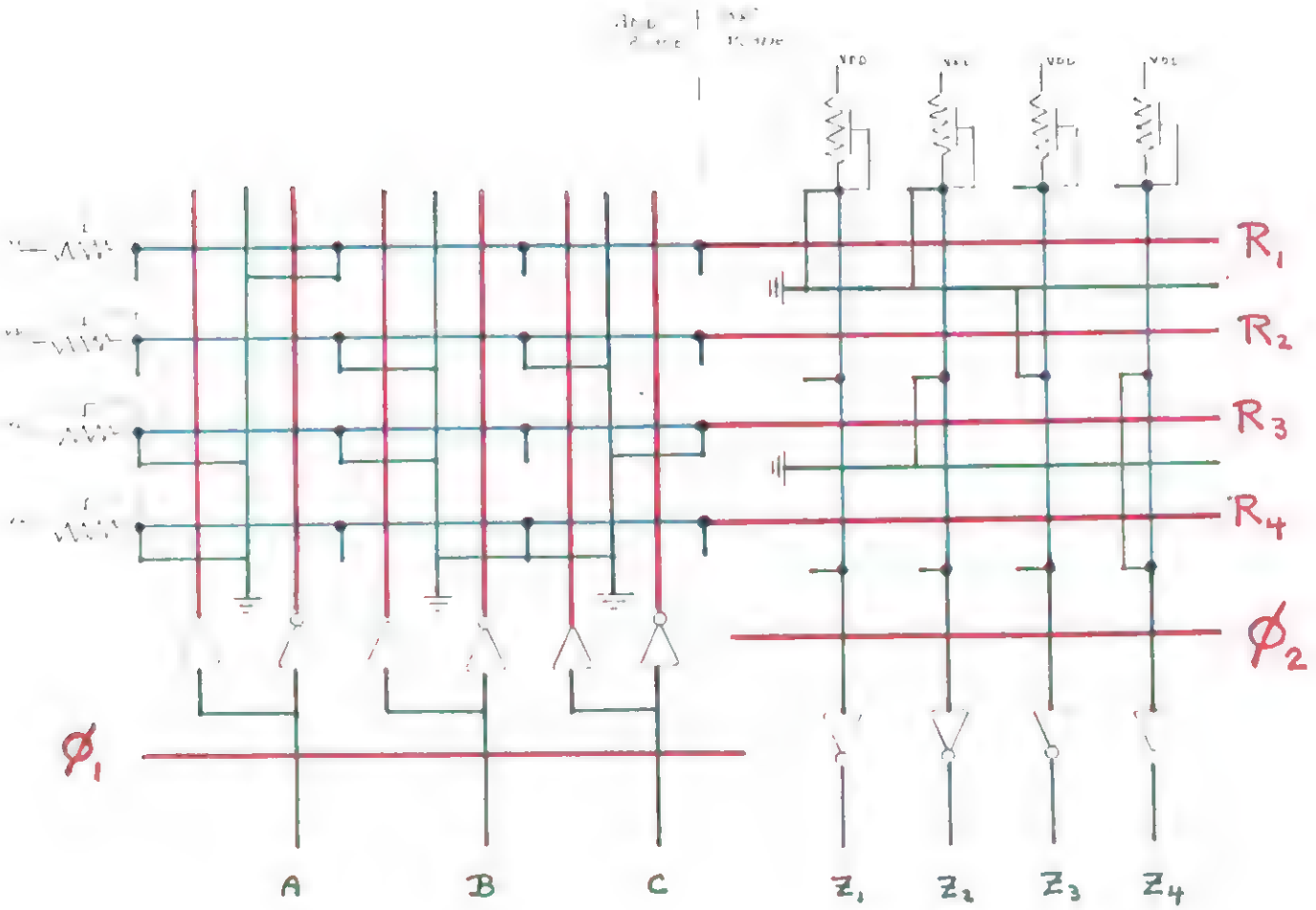


Fig.13c. Stick Diagram of PLA Example

Product Terms:

$R_1 = (A')' = A$

$R_2 = (B+C)' = B'C'$

$R_3 = (A+B+C')' = A'B'C$

$R_4 = (A+B'+C)' = A'BC'$

Outputs:

$Z_1 = A$

$Z_2 = A + A'B'C$

$Z_3 = B'C'$

$Z_4 = A'B'C + A'BC'$

Finite State Machines

In many cases in the processing of data, it is necessary to know the outcome of the current processing step before proceeding with the next. Results of the current step may be used as inputs in the next step. The configuration shown in figure 14a can be used to implement a processing stage having this requirement. A typical register to register transfer stage has been modified by simply feeding back some of its outputs to some of its inputs. This structure implements a form of sequential machine known as a *finite state machine.*

The feedback signals form a binary number which may be regarded as identifying the *state* of the machine. The value of this number is stored, along with the external inputs, in the first register during φ_1. These combined inputs then propogate through the combinational logic. The resulting outputs are stored in the second register during φ_2. The falling edge of φ_2 must occur a sufficient time later to insure that all signals have propagated throught the combinational logic. Each φ_1 - φ_2 cycle results in two new sets of outputs: (i) the external outputs which are typically used for controlling other units of the system, and (ii) a new feedback number, which defines the *next state* of the machine. This process repeats during each clock period. The number of possible states is determined by the number of bits in the feedback path, and is *finite.*

There are a number of ways of abstractly representing the states, the required state transitions, and the outputs of sequential machines under given input sequences. Possible representations include state diagrams, transition tables, boolean difference equations, etc. A large body of theory has been developed concerning the synthesis and analysis of sequential machines. The serious reader will benefit from a further study of the results of switching theory on this subject (R3, R4).

Implementations of simple finite state machines are used to produce the very lowest level of system control sequencing, since they can autonomously generate control sequences. The sequential machine having a finite number of states is a very important element in the hierarchy of fundamental concepts used in integrated system architecture.

The configuration shown in figure 14a implements a *synchronous* machine, since the feedback loop is only activated at times determined by the clock signals. In any clock period k, the output terms Z_j and the next state terms Y_f are valid during $\varphi_1(k)$. They are functions of the external inputs X_i and feedback terms Y_f which were valid during $\varphi_1(k-1)$.

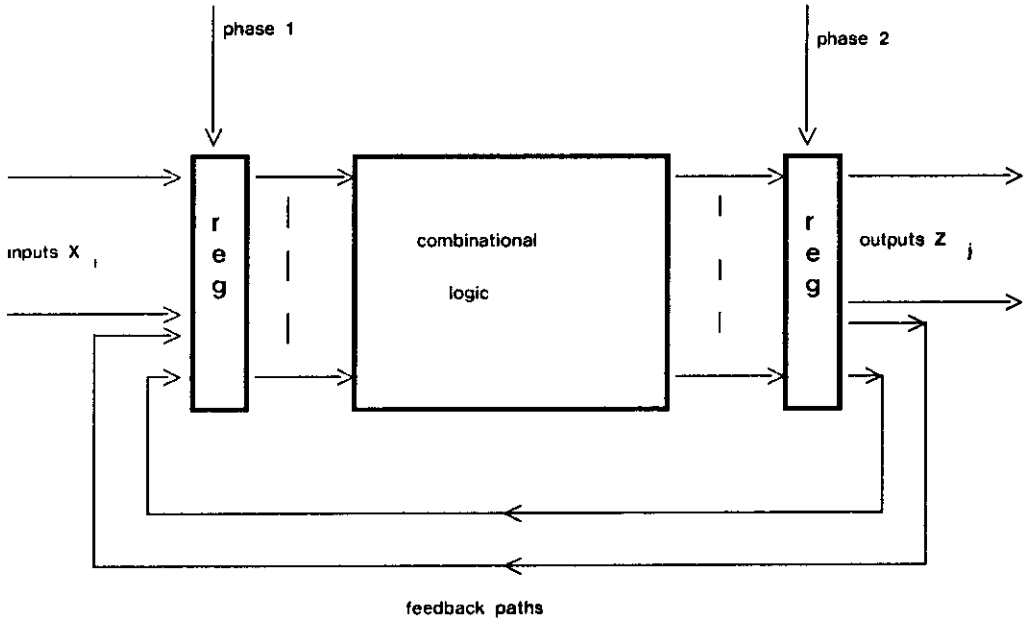


Fig. 14a. Feedback in Register Transfer Path, Implementing a Finite State Machine

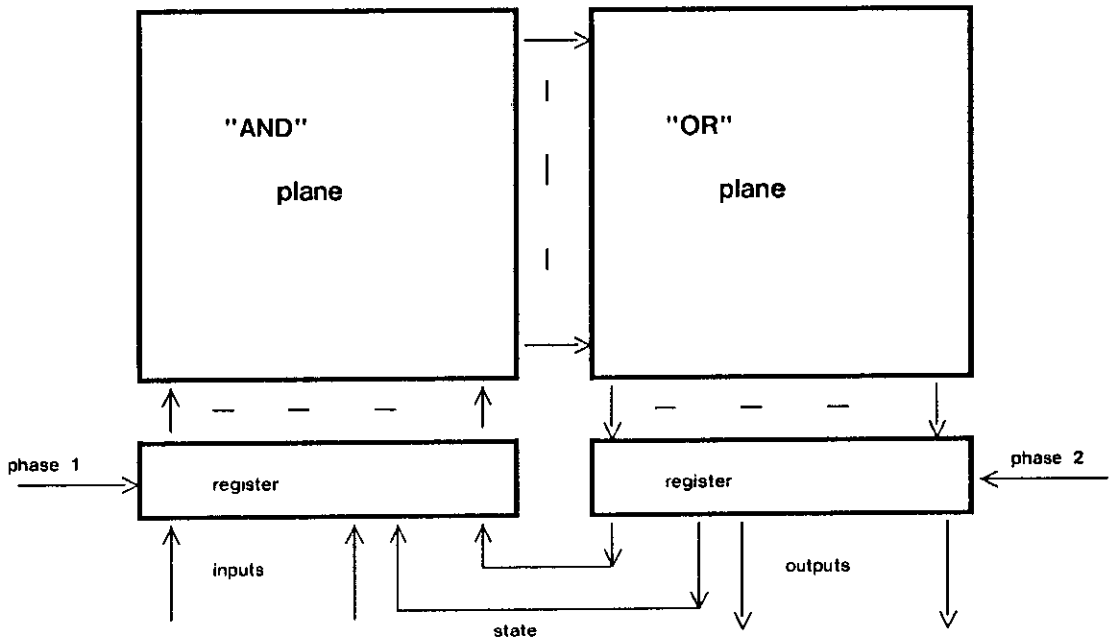


Fig. 14b. PLA Implementation of a Finite State Machine

If a sequential machine contains a feedback loop which is continuously active, then it may begin a response to a change in inputs or state at any time, rather than just at fixed clock times. Such a sequential machine is referred to as an *asynchronous sequential machine.* The analysis of asynchronous sequential machines and their implementation is far more complex than that of synchronous ones. Great care must be exercised to avoid any difference in state sequencing and outputs under arbitrary differential delays of signals through the circuit paths of such machines (R3. Ch.5). There will be only a few special cases where we use the asynchronous form of sequential machine (Chapter 7), and these will be subject to detailed analysis.

Where sequential machines are required within integrated systems, we will generally implement them in synchronous form. Synchronous machines are rather easy to implement correctly, and fit naturally into the two phase clocking scheme used for moving data around within our systems.

However, the reader should carefully note that an implementation of a synchronous sequential machine functions correctly only if the delays in the circuit paths are sufficiently short compared to the clock period. If we were to implement many copies of a particular machine, the probability of correct function for any given copy is thus a function of (i) the clock period used, and (ii) the distribution of differential delays in that copy's signal paths. Our estimate that a particular copy will function correctly is thus based in part on assumptions about the ratio of likely deviations in circuit delays to the clock period. A discussion of delays in MOS circuits is given in chapter 1.

There is a very straightforward way to implement simple finite state machines in integrated systems: we use the PLA form of combinational logic and just feed back some of the outputs to the inputs, as illustrated in figure 14b. The circuit's structure is topologically regular, has a reasonable topological interface as a subsystem, and is of a shape and size that are functions of the appropriate parameters. The function of this circuit is determined by the "programming" of its PLA logic. If, for example, early in a design cycle there is some uncertainty in the details of the desired sequencing of such a circuit, it is easy to provide layout space for extra, unused inputs, minterms, or outputs as contingencies.

The following simple example will help illustrate the basic concepts of finite state machines and their implementation in nMOS circuitry. A busy highway is intersected by a little used

farmroad, as shown in figure 15a. Detectors are installed which cause the signal C to go *high* in the presence of a car or cars on the farmroad at the positions labelled C. We wish to control traffic lights at the intersection, so that in the absence of any cars waiting to cross or turn left on the highway from the farmroad, the highway lights will remain green. If any cars are detected at either position C, we wish the highway lights to cycle through caution to red, and the farmroad lights then to turn green. The farmroad light is to remain green only while the detectors signal the presence of a car or cars, but never longer than some fraction of a minute. The farmroad light is then to cycle through caution to red, and the highway light then to turn green. The highway light is not to be interruptible again by the farmroad traffic until some fraction of a minute has passed.

A state diagram model of a finite state machine to control the lights is sketched in figure 15b. This diagram identifies four possible states of the machine, and indicates the input conditions which cause all possible state transitions. A block diagram of the PLA circuit implementing the machine is shown in figure 15c. The circuit uses the signal C as an input, and provides outputs HL and FL which encode the colors of the highway and farmroad lights it controls. Note that a timer is used to provide, as controller inputs, the short and long timeout signals (TS, and TL), at appropriate times following a start timer (ST) signal output from the controller. This timer could be implemented as a digital counter in the same nMOS circuitry. Another abstract model describing the desired function of the controller is given in the state transition table in figure 15d, which contains similar information to that in the state diagram.

The detailed sequencing of the machine under various input sequences is described by both the state diagram and transition table models of the controller. Consider starting in the state HG, where the highway lights are green. The machine remains in state HG as long as either no cars are detected or the long timeout has not occurrred, in other words as long as (C)AND(TL) = 0. After the long timeout occurs, if any cars are detected, the machine restarts the timer and changes state to HY, where the highway lights are yellow. It remains in state HY only until the short timeout occurs, and then restarts the timer and changes to state FG, where the farmroad lights are green. It remains in state FG until either no cars are detected or the long timeout occurs, i.e. (C)'OR(TL) = 1. Then it restarts the timer and changes to state FY, where the farmroad light is yellow. It remains in state HY only until the short timeout occurs. It then restarts the timer and changes to state HG, the starting state.

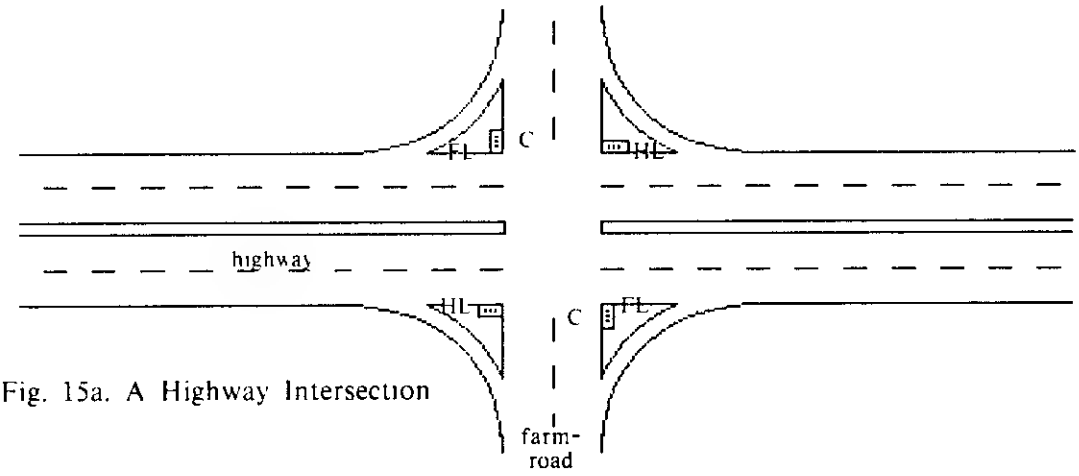


Fig. 15a. A Highway Intersection

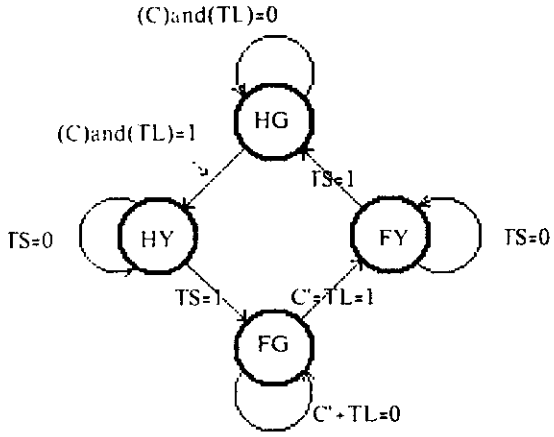


Fig. 15b. Light Controller State Diagram

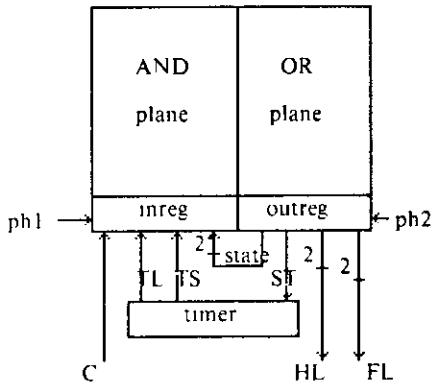


Fig. 15c. Controller Block Diagram

In Present State:	If Inputs are *	Next State will be:	and Outputs are: HI	FL	ST
Highway Green	(Cars)and(TimeoutL) = 0	Highway Green	Green	Red	No
	(Cars)and(TimeoutL) = 1	Highway Yellow	Green	Red	Yes
Highway Yellow	TimeoutS = 0	Highway Yellow	Yellow	Red	No
	TimeoutS = 1	Farmroad Green	Yellow	Red	Yes
Farmroad Green	(Cars)'or(TimeoutL) = 0	Farmroad Green	Red	Green	No
	(Cars)'or(TimeoutL) = 1	Farmroad Yellow	Red	Green	Yes
Farmroad Yellow	TimeoutS = 0	Farmroad Yellow	Red	Yellow	No
	TimeoutS = 1	Highway Green	Red	Yellow	Yes

Fig. 15d. Transition Table for the Light Controller

* Inputs not listed = don't cares

The locations of transistors in the PLA light controller circuit can be determined by "hand assembling" the "program" specified in the "symbolic" transition table in figure 15d, resulting in the encoded state transition of figure 15e. First we assign codes to the states. In the example: state HG is encoded as $(Y_0,Y_1) = (0,0)$; HY as (0,1); FG as (1,1); and FY as (1,0). Next, we assign codes to the output light control signals: green is encoded as (0,0), yellow as (0,1), and red as (1,0). We now form the encoded state transition table by constructing one row for each product term implied by the original symbolic table of figure 15d. A row in table 15d specifying a state transition as a function of a single input variable or single product term of input variables produces a single row in table 15e. A row in table 15d specifying a state transition as a function of a sum or sum of products of input variables, leads to a corresponding number of rows in table 15e.

Placement of the transistors within the PLA matrices follows directly from the encoded state transition table:

(i) For each logic-1 in the next state and output columns in the table, we run a diffusion path *from* the corresponding next state or output line in the PLA OR-plane, *under* the corresponding product term line, *to* ground. This creates a transistor controlled by the product term line. Then, if that controlling product term line is ever *high*, the path to the output inverter will be *low*, and the output will be *high*. The output line will be *low* unless some product term line controlling it is *high*.

(ii) For each logic-1 in the input and present state columns in the table, we run a diffusion path *from* the corresponding product term line, *under* the corresponding *inverted* input or next state line in the PLA AND-plane, *to* ground. The transistor thus created is controlled by the *inverted* input or next state line. Unless that controlling line is *high*, the product term line will be *low*.

(iii) For each logic-0 in the input and present state columns in the table, we run a diffusion path *from* the corresponding product term line, *under* the corresponding *non-inverted* input or next state line in the PLA AND-plane, *to* ground. The transistor thus created is controlled by the *non-inverted* input or next state line. Unless that controlling line is *high*, the product term line will be *low*.

Note that if all lines which control the transistors connecting a given product term line to ground are *low*, then that product term line will be *high*. Otherwise it will be *low*.

Stored during φ_1 in INREG *Stored during φ_2 in OUTREG*

Inputs:			*Present State:*	*Next State:*	*Outputs:*					*Product terms:*
C	TL	TS	Y_{p0},Y_{p1}	Y_{n0},Y_{n1}	ST	HL_0	HL_1	FL_0	FL_1	
0	X	X	0, 0 (HG)	0, 0 (HG)	0	0	0	1	0	R1
X	0	X	0, 0 (HG)	0, 0 (HG)	0	0	0	1	0	R2
1	1	X	0, 0 (HG)	0, 1 (HY)	1	0	0	1	0	R3
X	X	0	0, 1 (HY)	0, 1 (HY)	0	0	1	1	0	R4
X	X	1	0, 1 (HY)	1, 1 (FG)	1	0	1	1	0	R5
1	0	X	1, 1 (FG)	1, 1 (FG)	0	1	0	0	0	R6
0	X	X	1, 1 (FG)	1, 0 (FY)	1	1	0	0	0	R7
X	1	X	1, 1 (FG)	1, 0 (FY)	1	1	0	0	0	R8
X	X	0	1, 0 (FY)	1, 0 (FY)	0	1	0	0	1	R9
X	X	1	1, 0 (FY)	0, 0 (HG)	1	1	0	0	1	R10

Fig.15e. Encoded State Transition Table for the Light Controller

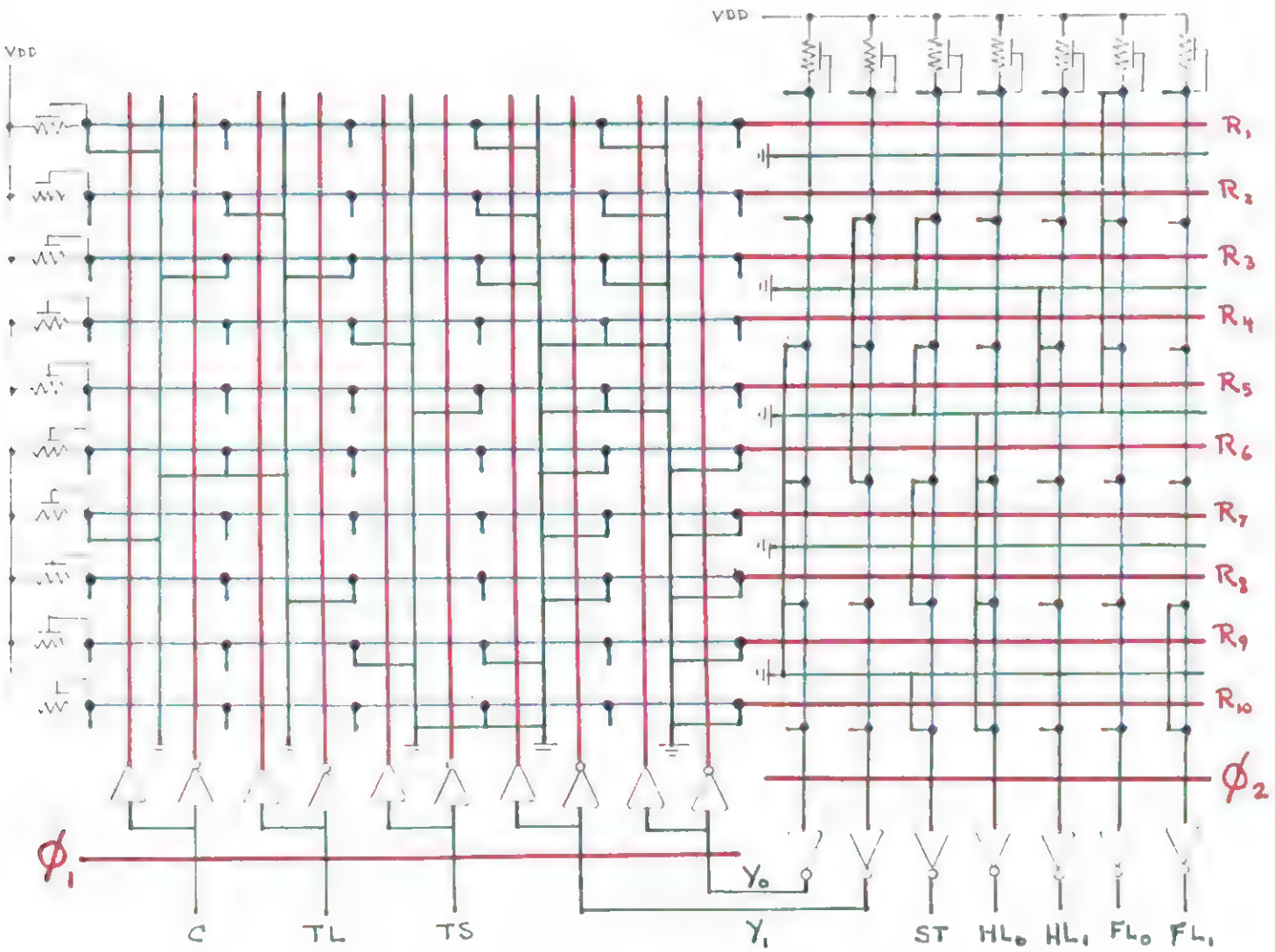


Fig.15f. PLA Sequential Circuit Implementing the Light Controller

The PLA circuit in figure 15f is programmed from the transition table in figure 15e, according to the rules above, and implements the traffic light controller. Note that this LSI implementation does not exactly strain itself to meet the time response requirements of the control problem: it can run at a clock rate at least 10^7 times as fast as required. Also, note that the PLA controller is roughly $(150\lambda)^2$ in area. Using the 1978 value of $\lambda = 3\mu m$, this controller is $(450\mu m)^2 \sim 0.002$ cm^2 in area. A PLA controller this size may contain over 150 transistors, but occupies only $1/125^{th}$ of the area of a typical 0.25 cm^2 silicon chip in 1977. By the late-80's, as λ scales down towards its ultimate limits, such a controller will require only $\sim 1/25{,}000^{th}$ of the area of such a chip.

As we will see in later chapters, a data processing machine of any desired complexity can be created by interconnecting register to register data processing paths constructed along the lines of that shown in figure 11c, such paths being controlled by finite state machines implemented as shown in figure 14b. The data paths form the "highways" for the movement of data, under control of the finite state machine "traffic controllers".

Towards a Structured Design Methodology

The task of designing very complex systems involves managing, in some highly structured way, the space and time relationships between the various levels of system building blocks so that the entire system will function as intended when it is finished. The beginnings of a structured design methodology for VLSI systems can be produced by merging together in a hierarchy the concepts presented in this chapter. Designs are then done in a "top down" manner, but with a full understanding by the architect of the successive lower levels of the hierarchy.

To begin, we plan our digital processing systems as combinations of register to register data transfer paths, controlled by finite state machines. Then the geometric shapes, relative sizes, and interconnection topologies of all subsystem modules are collectively planned so all modules will merge together snugly, with a minimum of space and time wasted by random interconnect wiring. Storage registers are typically constructed by using charge stored on input gates of inverting logic. The combinational logic in the data paths is typically implemented using steering logic composed of regular structures of pass transistors. Most of the combinational logic in the finite state machines is typically implemented using PLA's. All functioning is sequenced using a two-phase, non-overlapping clock scheme.

When viewed in its entirety, a system designed in this manner is seen as a hierarchy of building blocks, from the very lowest level device and circuit constructs, on up to and including the high level system software and application programs in which the intended functions of the system are finally expressed. Individuals who understand the key concepts of each level in this hierarchy will recognize that the boundaries between levels are rather elastic ones. Each level of activity might best be optimized not on its own as a specialty, but as it fits into an overall systems picture. For example, the activity "logic design" in integrated systems might best be conceptualized as the search for techniques and inventions which best couple the physical, topological, and geometric properties of integrated devices and circuits with the desired properties of digital VLSI systems. The search for alternative components for any given design hierarchy, and the search for alternative hierarchies, will be done best by those who span more than one specialty.

A particularly uniform view of such a system of nested modules emerges if we view every module at every level as a finite state machine or data path controlled by a finite state machine. At the lowest level, elements such as the stack and register cells may be viewed as

state machines with one feedback term (the output), two external inputs (the control signals), and a one bit state register. These rudimentary state machines are grouped in a structured manner to form portions of a state machine, or data path controlled by a state machine, at the next level of the hierarchy. Structured arrays of identical state machines often provide a mechanism for distributing processing among memory cells[R6], thus enabling vast increases in processing bandwidth. Although in some cases the feedback paths are used in rather specialized ways, the state machine metaphor still provides a precise description of module behavior. The entire system may thus be viewed as a giant hierarchy of nested machines, each level containing and controlling those below it. A detailed quantitative treatment of certain hierarchically organized machines is given in chapter 9.

In chapters 5 and 6 we will apply the design methodology developed in this chapter to the design of a digital computer system. A one chip implementation of the data path portion of this computer system is illustrated in the frontispiece. Consistent use of the described design methodology resulted in a design of great regularity, short delay times, low power consumption, and high logical processing capability. As we will see in chapter 4, regular designs, with small numbers of basic circuit cell types replicated in two dimensions to form subsystems, also have significant implementation advantages over less structured designs.

References

1. - - - *polycell ref* - - -

2. I. E. Sutherland, C. A. Mead, "Microelectronics and Computer Science", *Scientific American*, September 1977, pp. 210-228.

3. J. D. Williams, "Sticks -- A New Approach to LSI Design", M.S.E.E. Thesis, Dept. of Electrical Engineering, M. I. T., June, 1977.

4. W. M. Penney, L. Lau, Eds., "MOS Integrated Circuits", Van Nostrand, 1972. Chapter 5.

5. C. H. Sequin, M. F. Tompsett, "Charge Transfer Devices", Academic Press, 1975, Ch. VIII.

Reading References

R1. C. G. Bell, A. Newell, "Computer Structures: Readings and Examples", McGraw-Hill, 1971, contains an excellent discussion of the levels in the hierarchy of computer architecture, and many specific examples of computer structures.

R2. B. Soucek, "Microprocessors and Microcomputers", John Wiley, 1976, is a good introductory reference containing sections on basic digital design, and on the interfacing and programming of a number of present day microprocessors.

R3. D. L. Dietmeyer, "Logic Design of Digital Systems", Allyn and Bacon, 1971, is a comprehensive text on switching theory and logic design.

R4. Z. Kohavi, "Switching and Finite Automata Theory", McGraw-Hill, 1970, is another good text on switching theory.

R5. S. H. Caldwell, "Switching Circuits and Logical Design", John Wiley, 1958, is an early text containing interesting material on relay contact networks.

R6. S. H. Unger, "A Computer Oriented Towards Spatial Problems", Proc. of IRE, vol. 46, no. 10, pp. 1744-1750, Oct. 1958, is an early paper describing a spatially distributed processor, anticipating present strategies for distributing processing into memory.

R7. C. E. Shannon, "Symbolic Analysis of Relay and Switching Circuits", *Trans. of AIEE*, Vol. 57, 1938, pp. 713-723, is the classic paper proposing a method for the mathematical treatment of switching circuits.

R8. G. Boole, "An Investigation of the Laws of Thought", London, 1854, reprinted by Dover Publications, contains a presentation of the algebra of logic on which Shannon based his switching algebra.

Chapter 4: Implementing Integrated System Designs: From Circuit Topology to Patterning Geometry to Wafer Fabrication

Sections:

This chapter presents the basic concepts involved in implementing integrated system designs, from the system designer's point of view. Tools are described which help the designer produce the geometrical layout patterns for each layer of an integrated system given the logic, circuit, or topological level design of the system. Procedures are described for encoding these layout patterns and then using the encoded layouts in the patterning and fabrication processes to implement the integrated system. In addition, we discuss how design tools and procedures are likely to evolve towards fully integrated design systems, under the influence of increased complexity of design and predictable changes in the technologies of implementation.

To enable groups of readers to actually design moderate sized LSI systems, we've included descriptions of easily constructed LSI design tools and procedures for organizing and implementing LSI multi-project chips. In each case, the tools are described as part of a complete system of design and implementation procedures, some of which are performed manually while others are machine assisted. Those experienced in software system design will recognize that construction of the machine-assisted portions of these systems is fairly straightforward. Contrary to what many may think, designing your own LSI projects, merging them onto collaborative multi-project chips, and having these implemented by commercial maskmaking and wafer-fabrication firms is now well within the computational and financial reach of most industrial R&D groups and university EE/CS departments.

We are firm believers in *learning by doing*, and hope that the information provided in this chapter will help and encourage many groups of readers to try their hand at building LSI design tools and designing LSI systems. Such first-hand experience will lead to a deeper understanding of the remaining material in this text.

An overview of the stages of integrated system design, layout, and implementation is given in figure 1. The designer first transforms the circuit and topological level designs into a geometrical layout of the system, using procedures described later in this chapter. In order to optimize the layout, perform various design checks, and discover errors, the designer usually iterates several times between *design* and *layout*. The result is a set of *design files* describing the layout. These files are in a particular representation called an *intermediate form*, which efficiently and unambiguously describes the layout geometry.

The design files are then converted into files for driving the chosen patterning mechanism. At present, design files are commonly converted into *pattern generator* (PG) files, for use by a *maskmaking* firm for driving an optical pattern generator, the first step of maskmaking. By a sequence of photolithographic steps, the mask house produces a set of *masks*, which a commercial *wafer fabrication* firm may then use to pattern silicon *wafers*. Each finished wafer contains an array of system chips. The wafers are then diced into separate chips, which are packaged and tested to yield working systems.

From the system designer's point of view, maskmaking and fabrication can be visualized as one would a film processing service: the designer produces the "artwork" (design files), from which the mask house makes "negatives" (masks), which are then run on a fab line to produce "prints" (wafers). The maskmaking and fabrication sequence is *function, design, and layout independent*: the mask and fab firms do not require detailed information about the integrated systems they fabricate. If the original layouts satify the design rules, and satisfy a few constraints imposed by patterning and fabrication, then these processes will yield correctly patterned wafers.

One need not closely bind a system's design to the detailed processing specifications of particular mask and fab firms. Various firms will differ somewhat in the minimum value of the length unit λ which they can successfully process. The transit time of the transistors fabricated, and the resistance per square and capacitance per unit area of fabricated features will also vary from one fab line to another. However, well structured and relatively process independent nMOS designs will function correctly if scaled to a value of λ appropriate for the chosen fabrication facilities, and operated using an appropriate system clock period.

We next examine some of the present implementation procedures a bit more closely, to set the stage for sections on design and layout. Those later sections will be clearer if one can visualize how the design files are to be used during patterning and fabrication.

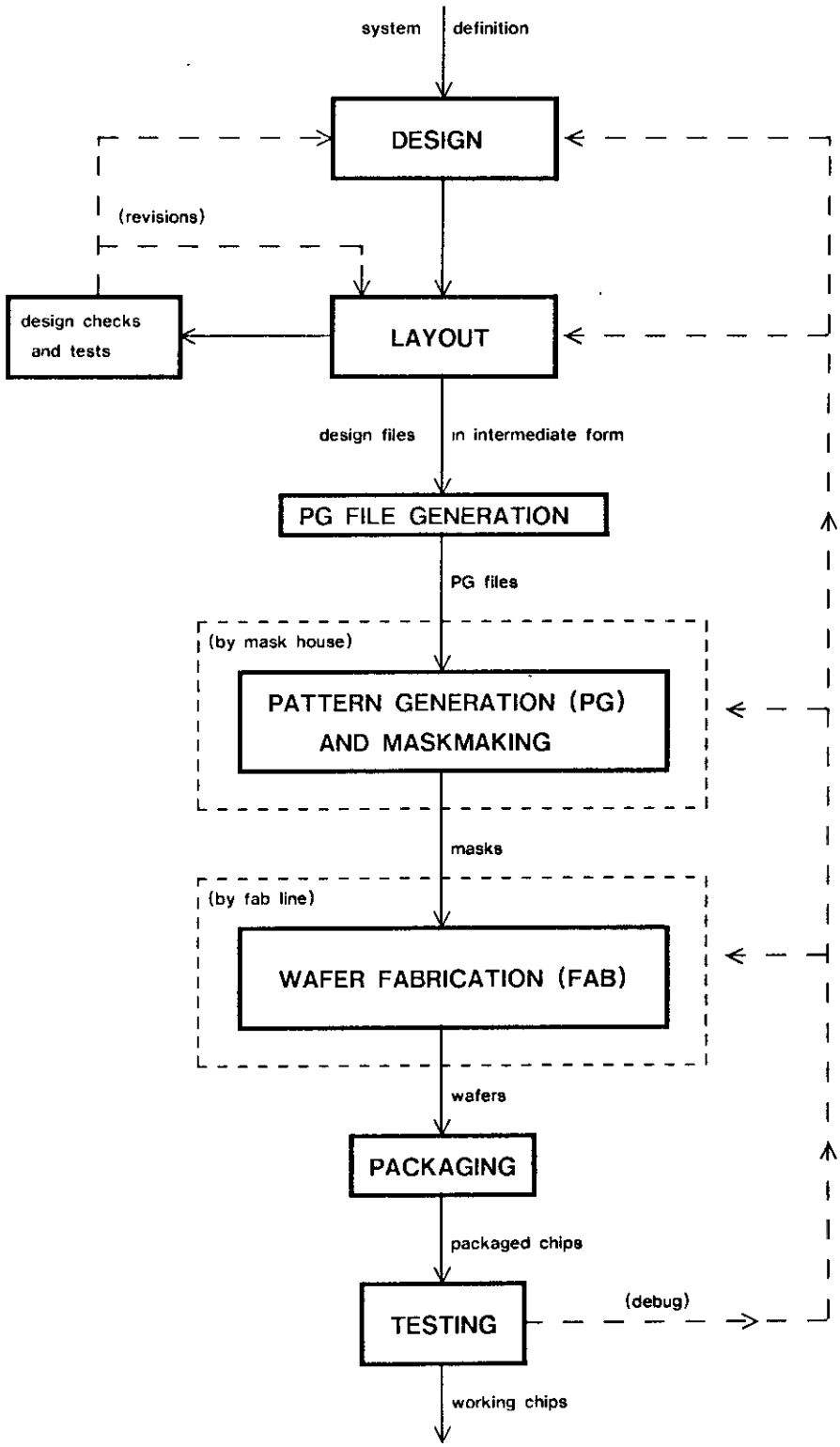


Fig. 1. Overview of Integrated System Implementation

(overview sil)

Patterning and Fabrication

On completion of design and layout, the system design is contained in *system layout files in intermediate form*. Prior to fabrication, a final *check plot* of the layout is usually generated by converting these design files into files for driving a graphics plotter. Check plots are used for visually checking for design rule violations and other design errors. Once the designers have done as much visual checking as they are going to do, the system layout files are converted into *pattern generator* (PG) *files*, to be sent to the maskmaking facility. Figure 2 summarizes the sequence of patterning and fabrication procedures which then follows, and identifies the artifacts passed on at each step in the sequence.

Maskmaking begins with *pattern generation* to produce *reticles*. Present pattern generators are projector-like systems containing (i) a precisely movable stage, (ii) an aperture of precisely variable rectangular size and angular orientation, and (iii) a light source, all program controllable by a computer system. To produce a reticle, a photographic plate is mounted on the stage, and the PG file for a particular system layer is used to direct the flashing of a sequence of rectangular exposures, of particular sizes and orientations, onto a sequence of coordinate locations on the plate, as illustrated in figure 3.

The PG file contains a sequence of entries, each of which describes a rectangle. A typical representation uses five numbers for each rectangle: the x,y coordinates of its center, and its height, width, and angular orientation, as shown in figure 4. One can now visualize the nature of the conversion from intermediate form to PG files: the layout of each layer must be decomposed into its equivalent as a set of rectangles, each having [x,y,h,w,a] values flashable by the particular pattern generator, and these rectangles must be sorted into an efficient flashing sequence for that pattern generator.

When the flashing sequence is completed, the plate is developed, yielding the reticle. A sketch of such a reticle is given in figure 5. Each reticle is a photographic master copy much like a photo negative, of the layout of one system layer, usually at a scale ten times (10x) the final system chip size. Photo enlargements of reticles, called *"blowbacks"*, may be obtained from the mask house, to provide a further level of checking of design layout, PG file conversion, and pattern generation. At the current value of $\lambda = 3$ microns, blowbacks at approximately 100 to150 times actual chip dimensions have sufficient detail to enable visual checking of the smallest features. Blowbacks of reticles may also be obtained in the form of

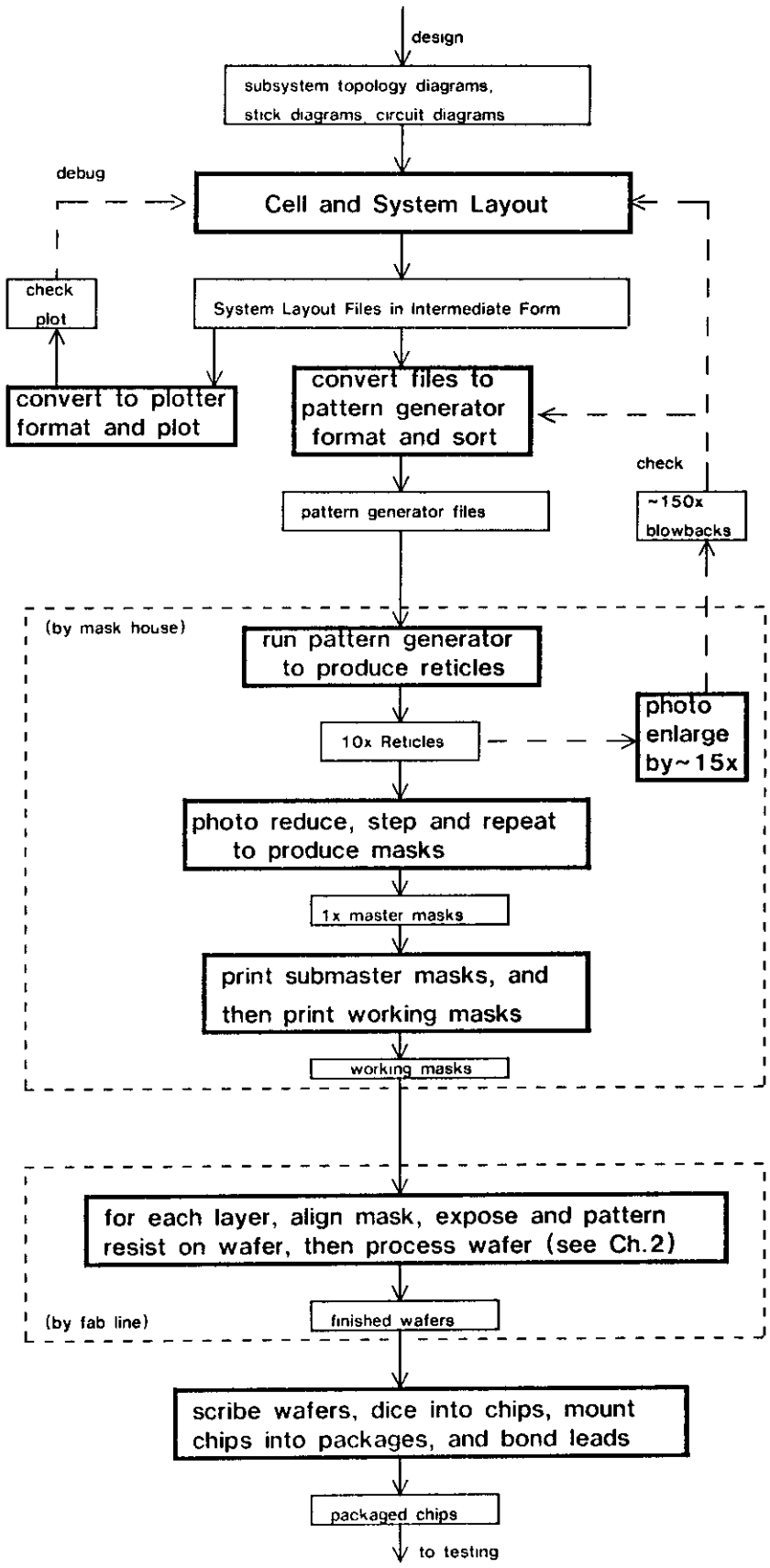


Fig. 2. The Procedures and Artifacts of the Typical Present Implementation Process

(proart sil)

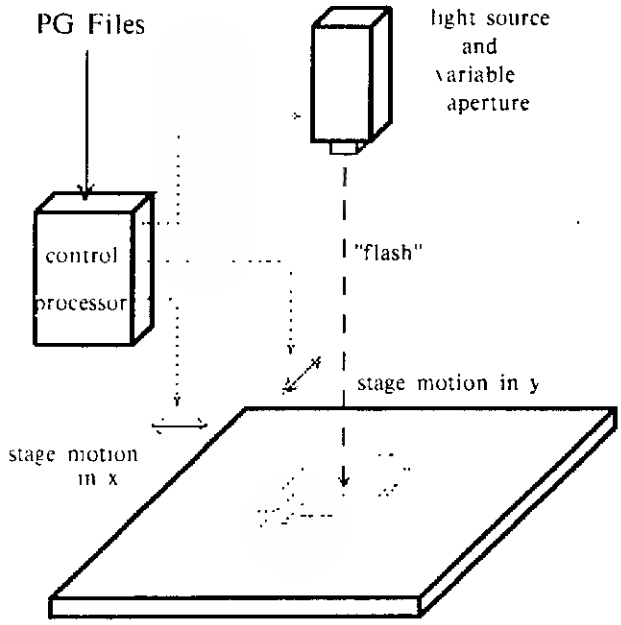


Fig. 3. Sketch Illustrating Function of Pattern Generator

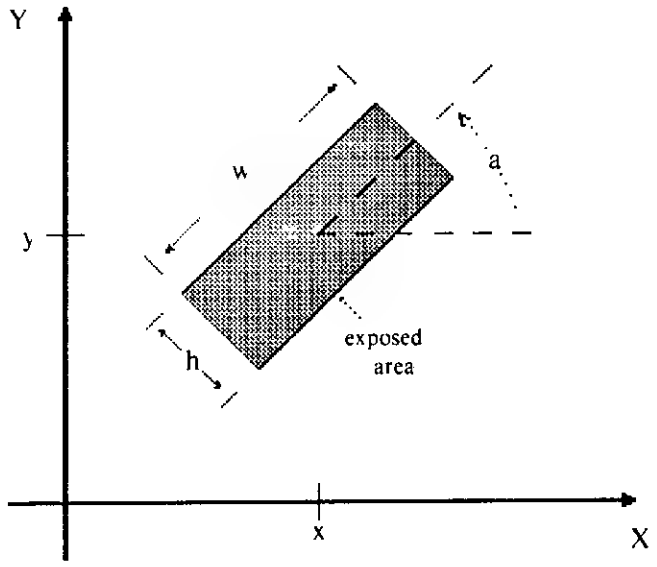


Fig. 4. Parameters of One Flash of Pattern Generator

(pg press)

color transparancies, to enable inspection of superposed overlays of various layers.

Once the 10x reticles have been generated, a *1x master mask* is made from each reticle using a *photorepeater*, often called a *step and repeat camera*. The photorepeater exposes a photographic plate held on a moveable stage, as in the pattern generator. In this case, however, each plate exposure is a 10:1 photo reduction of the reticle pattern. Between exposures the stage is moved by a precise x,y stepping distance. This process is repeated until a complete array of 1x chip patterns for one layer of the system has been exposed. The plate is then developed to produce a 1x master mask. Figure 6 sketches such a mask made from the reticle in figure 5.

Note that when each reticle is inserted in the photorepeater, the position and angular orientation of the reticle pattern is carefully adjusted by microscopic examination of two *fiducial marks* on the reticle. These marks are placed as part of the pattern generation process, and have the same precise position relative to the chip pattern origin on each of the system's reticles, thus assuring that all mask levels produced with the photorepeater will accurately register with each other.

A succession of contact prints is made from each master mask to yield a number of *working masks*, sometimes called working *plates*, for each system layer. These are the actual masks used in wafer fabrication. During the contact printing step of the typical wafer fabrication procedure, the working plates occasionally become worn or damaged, so several are usually made for each layer.

The wafer fabrication facility uses the working plates in the sequence of patterning and process steps described in chapter 2, to produce finished wafers. The fab line requires no detailed information about the design or mask patterns of the integrated system being fabricated. However, several auxiliary patterns are normally included in the mask patterns, some of which are replicated on each chip and are examined during wafer fabrication: (i) *alignment marks*, which are used to accurately overlay successive masks with previous patterning steps, (ii) *line width testers*, sometimes called *critical dimensions* (C/D's), which are lines in each mask layer of stated width that may be examined during maskmaking and fabrication to control dimensional tolerances, and (iii) a few simple *test transistors* and their associated probe pads, which may be electrically tested prior to packaging to verify that the wafer fabrication process was successful.

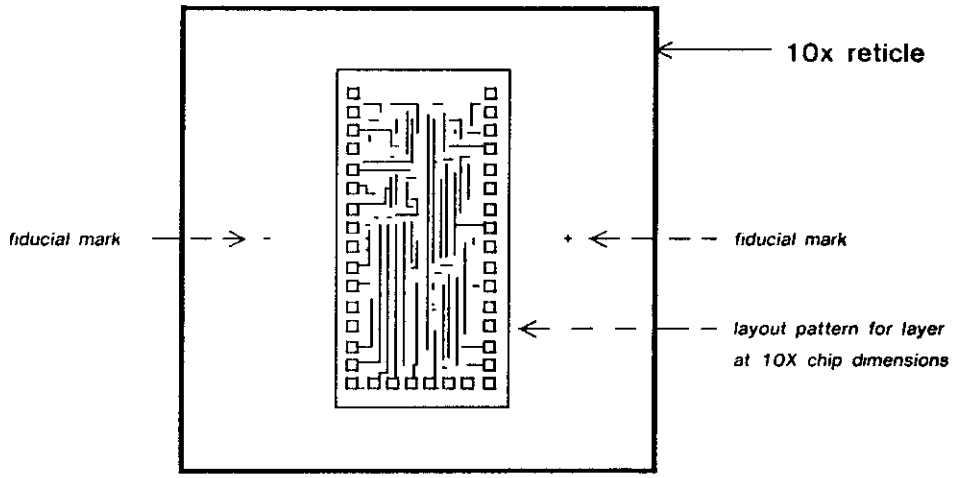


Fig. 5. Sketch of a 10x Reticle

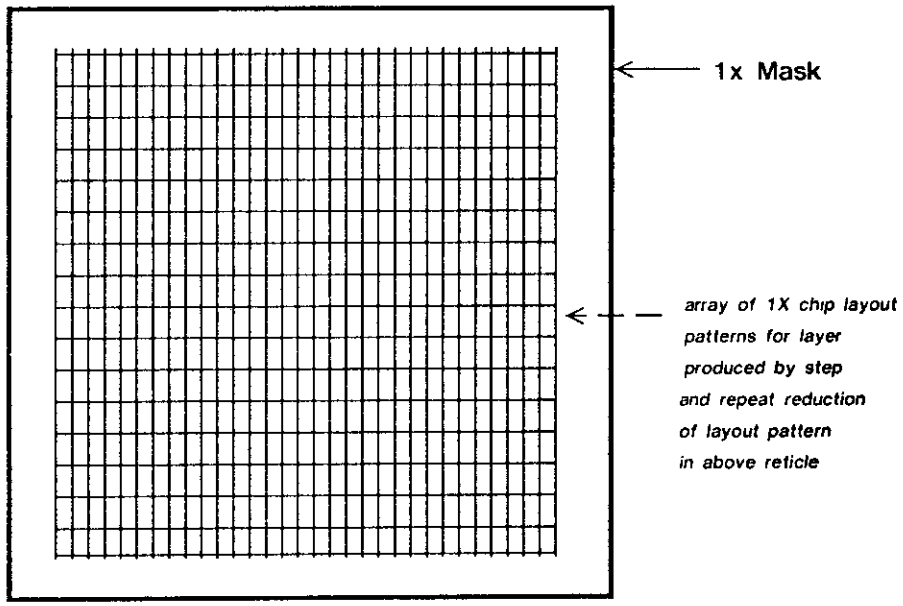


Fig. 6. Sketch of a Mask made from the above Reticle

The finished wafers are divided into chips and packaged by the sequence of steps sketched in figure 7. The wafers are diced into individual chips by first scribing their surface along the boundary lines between chips, called *scribe lines*, with either a diamond tipped scribe or a diamond edged saw blade, and then fracturing them along these lines. Each individual chip is then cemented into the cavity of a package. After fine wires are bonded between the contact pads on the chip and the leads of the package, and a cover cemented over the cavity, the system is ready for testing.

From the preceding we see that once a system's design files have been produced, all the remaining implementation procedures are design and layout independent, and largely automatic. However, the many extraneous parameters, patterns, and constraints involved in maskmaking and fabrication must be carefully thought through and defined in order to guarantee successful implementation within a reasonable turnaround time. The PG files must be correctly sorted and formatted for the pattern generator to be used. The 10x pattern of the chip must fit within the largest reticle that the pattern generator can produce. The photorepeater used will determine the shape, size, and location of the fiducial marks on the reticle. The size, surface material, and photographic polarity, either positive (*clear field-opaque features*) or negative (*dark field-transparent features*), of the working plates will be a function of the fabrication facility to be used. Each fab line also typically prescribes its own patterns for the alignment marks and test transistors to be included along with the system in the mask patterns.

While many designs may be scalable and have some longevity, the parameters, patterns, and constraints of maskmaking and fabrication are changing rapidly as the technologies evolve. This constant change complicates interactions with mask and fab firms. Later we describe procedures for implementating moderate sized LSI systems as part of multi-project chips. Such chips are collaborative efforts of many designers, enabling many projects to be merged into one maskmaking and wafer fabrication run. In this way the procedural overhead involved may be shared.

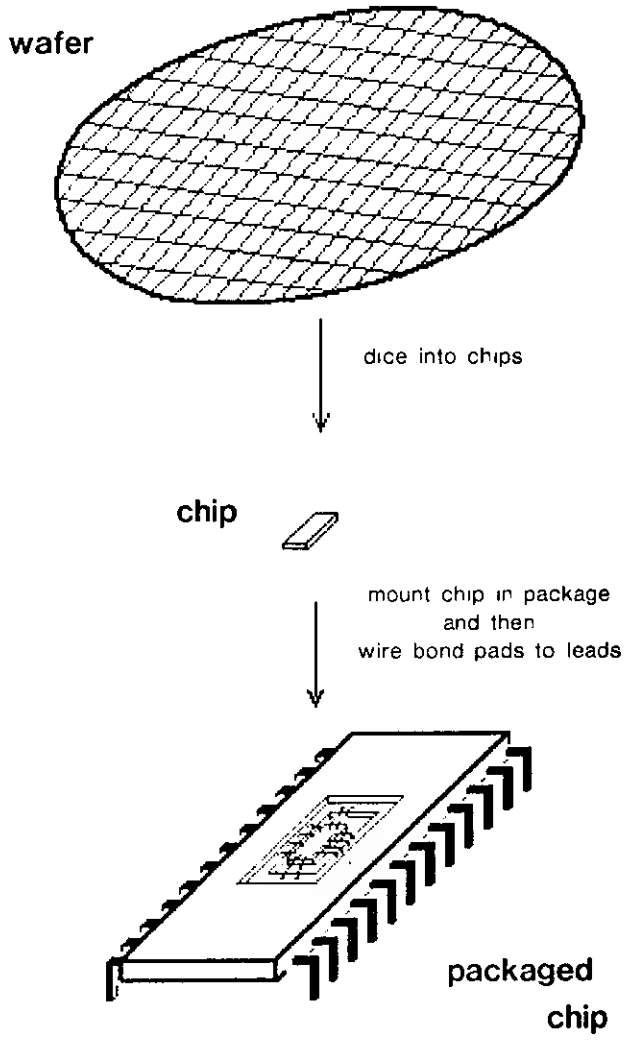


Fig. 7. Sketch Illustrating the Packaging Sequence

(wafer press)

Hand Layout and Digitization using a Symbolic Layout Language

A simple and common method of producing system layouts is to draw them by hand. This is typically done on a one lambda grid using the familiar color codes to identify various system layers. Once the layout has been hand drawn it can then be *digitized*, or translated into machine readable form, by encoding it into a symbolic layout language. Hand layout and digitization using a symbolic layout language is quite a practical method of generating design files for highly structured system designs. Be warned, however, that implementing irregular structures using these primitive procedures is a difficult and tedious task.

If a system has only a few cell types which are replicated over and over, and otherwise has little "random wiring", one need draw only a single copy of each cell type, and then make reproductions or equivalent sized outlines of these cell drawings. All these cell reproductions may then be patched together to plan and build up the overall layout. Similarly, only one symbolic digitization need be made for each cell type. The replication of cells in various orientations and locations in the system layout can then be easily described using the symbolic layout language. In a sense, the ease with which a system's layout can be described using a primitive layout languge provides a measure of the regularity of its design. The OM2 Data Chip pictured in the frontispiece was laid out and digitized in this way, using only the simplest machine aids.

The function of a symbolic layout language, in its simplest form, is similar to that of a macro-assembler. The user defines *symbols* (macros) which describe the layout of basic system cells. The locations and orientations of instances of these symbols are described in the language, as a function of appropriate parameters. These symbolic descriptions may then be mechanically processed in a manner similar to the expansion of a macro assembly language program, to yield the intermediate form description of the system layout, which is analogous to machine code for generating output files. An example intermediate form is described in a later section. The intermediate form files may be processed to yield the PG files: each layer being a machine encoded collection of rectangles encoded as [x,y,h,w,a] values. The generation of PG files is analogous to the loading and execution of machine code to produce output files: it is a process of "unrolling" and fully instantiating all symbol descriptions into a sequence and format suitable for a particular output device. Definition of simple layout languages and the construction of their assemblers is fairly straightforward. The reader may define and implement layout languages by using the macro assembler or higher level language facilities of any commonly available computer system (R1, R3).

The following example will clarify the concepts and procedures of hand layout and symbolic layout description: We wish to create an array of shift registers consisting of parallel horizontal rows of inverters coupled by clocked pass transistors, as in figure 5a., chapter 2. Figure 8a sketches the stick diagram of one row of the array. The entire array can be constructed from one basic cell containing an inverter, the pass transistor following it, VDD and GND buses crossing through on metal, and a clock line passing through on poly. Figure 8b shows a hand sketch of the layout of the basic shift register cell, SRCELL, on a 1λ grid, subject to the design rules given in Ch.2, Sect.2. Since the inverters are coupled by pass transistors, the inverter pullup/pulldown ratio is ~ 8:1 (see Ch.1., Sect.2.). Also, while the 4λ wide metal lines could be 1λ narrower in between the contact regions, this would not decrease the cell size. As an exercise, the reader might check for design rule violations, and also for ways of further shrinking the cell size.

The SRCELL layout shown in figure 8b is composed using only rectangles placed at orientations which are integer multiples of 90°. The illustrations and descriptions in this section are considerably simplified by the use of such constrained layout constructions, and yet still illustrate the general principles involved. Were completely arbitrary shapes used, the SRCELL could be made somewhat smaller and still satisfy the design rules. Interestingly, experience has shown that the simple extension of including rectangles at orientations which are integer multiples of 45° enables most cell layouts to reach within a few percent of the minimum area achievable using arbitrary shapes. There is a clear tradeoff here: the inclusion of increasingly complex geometrical objects in a layout will tend to reduce the minimum achievable layout area, but will also increase the computational complexity of the associated machine aids.

We can informally characterize a simple layout language by examining figure 9, which contains a description of the layout of an array of SRCELLs using such a language. The language describes layouts as collections of BOXes on various layers. BOX statements describe each of these boxes by specifying their layer, the X,Y coordinates of their lower left corner and then their length, LX, in the x direction, and LY, in the y direction. The use of a box corner to encode its location simplifies the encoding task. BOX statements may describe arrays of identical boxes, with the array's lower left corner origin at X,Y, by including optional parameters which specify the number NX and replication interval IX in the x direction, and NY and IY in the y direction. Dimensions are given in the length unit, λ. A SCALE statement defines the value of λ for this particular layout as λ = 3.0 microns.

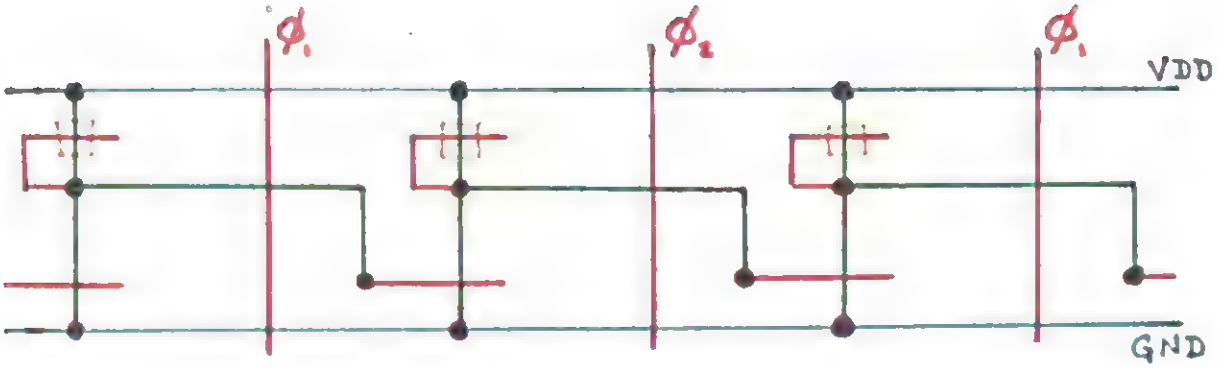


Fig. 8a. Stick Diagram of One Row of Shift Register Array

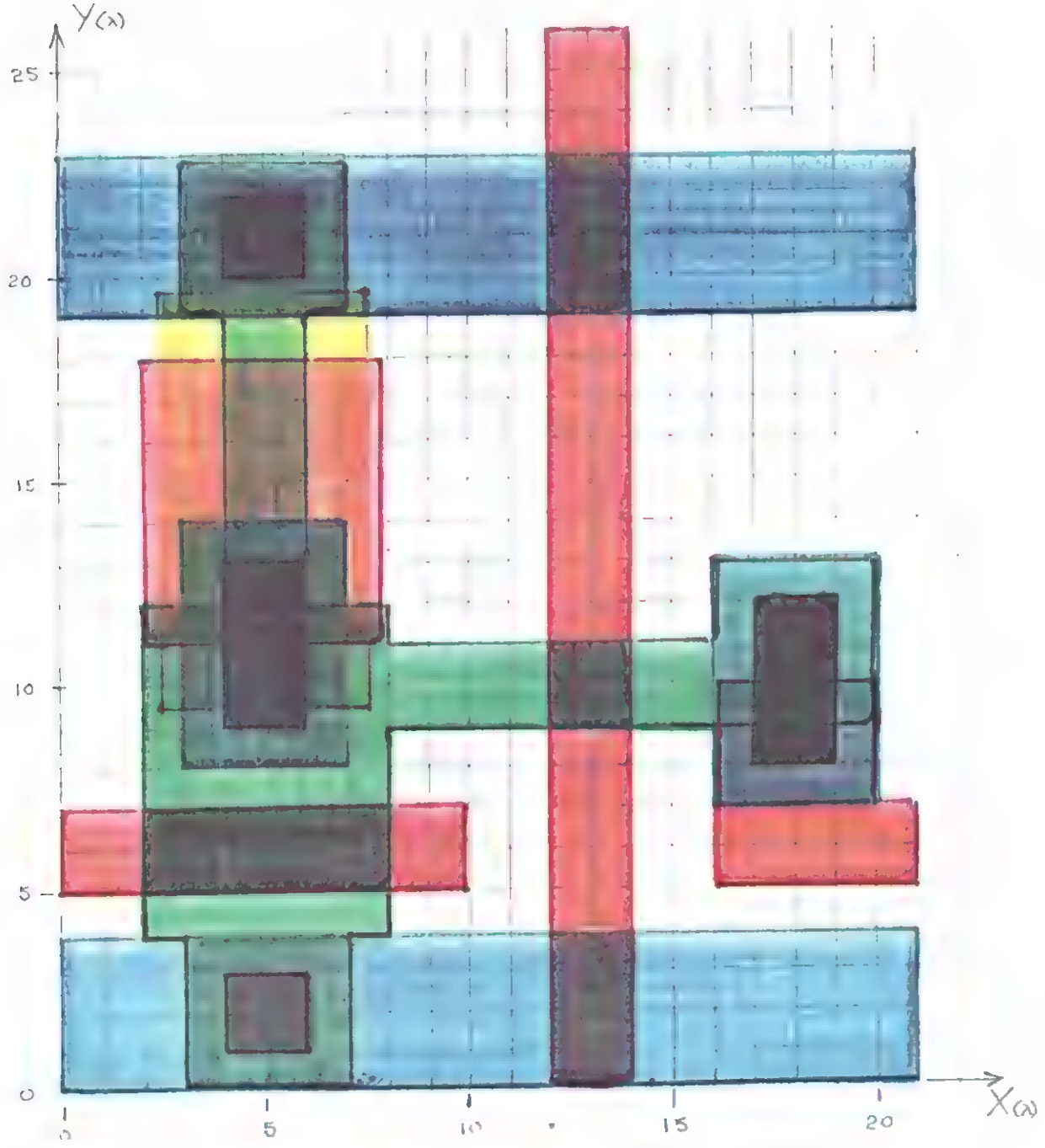


Fig. 8b. Hand Sketch of Layout of One Shift Register Cell

In figure 9, the SRCELL is first described as a macro, or SYMBOL. The reader can verify that the collection of BOXes in the definition of the SYMBOL SRCELL, when ORed together, produces the layout in figure 8b. This SRCELL is then replicated a number of times in various layout locations according to parameters in several DRAW statements.

Each DRAW statement describes the placement of an array of cells as follows: The cell described by the named SYMBOL definition is considered to be drawn at the origin. It is then *mirrored* (about the x and/or y axis), and/or *rotated* (by 0°, 90°, 180°, or 270°) about the origin, as specified by MIRROR or ANGLE transformations. The cell thus positioned may then be *replicated* NX times at distance intervals IX in the x direction, and that row of cells then be *replicated* NY times at intervals IY in the y direction. The resulting array of cells is then *translated* a distance X,Y from the origin, and placed into the layout.

```
SCALE    LAMBDA=3.0MICRON;
;
SYMBOL   START, SRCELL;
  BOX    DIFF,X=3,Y=0,LX=4,LY=4,NY=2,IY=19;
  BOX    DIFF,X=2,Y=4,LX=6,LY=8;
  BOX    DIFF,X=8,Y=9,LX=8,LY=2;              INVERTER OUTPUT
  BOX    DIFF,X=16,Y=9,LX=4,LY=4;
  BOX    DIFF,X=4,Y=12,LX=2,LY=7;
  BOX    IMPL,X=2.5,Y=9.5,LX=5,LY=10;         PULLUP IMPLANT
  BOX    POLY,X=0,Y=5,LX=10,LY=2;             CELL INPUT
  BOX    POLY,X=12,Y=0,LX=2,LY=26;            CLOCKLINE
  BOX    POLY,X=16,Y=5,LX=5,LY=4;             CELL OUTPUT
  BOX    POLY,X=16,Y=7,LX=4,LY=3;
  BOX    POLY,X=2,Y=11,LX=6,LY=7;
  BOX    CUTS,X=4,Y=1,LX=2,LY=2,NY=2,IY=19;
  BOX    CUTS,X=17,Y=8,LX=2,LY=4;
  BOX    CUTS,X=4,Y=9,LX=2,LY=4;
  BOX    METL,X=0,Y=0,LX=21,LY=4,NY=2,IY=19.  VDD & GND
  BOX    METL,X=3,Y=8,LX=4,LY=6;
  BOX    METL,X=16,Y=7,LX=4,LY=6;
SYMBOL   END;
;
  DRAW   SRCELL,NX=4,NY=2,IX=21,IY=38,X=0,Y=0;
  DRAW   SRCELL,MIRRORX,NX=4,IX=21,X=0,Y=42;
;
END;
```

Figure 9. Symbolic Description of Shift Register Array

The "program" in figure 9 describes an array of 3 rows and 4 columns of SRCELLs. After machine assembly of this program, the resulting design file can be used to generate check plots, which may be inspected to detect errors made in encoding the layout. A check plot of

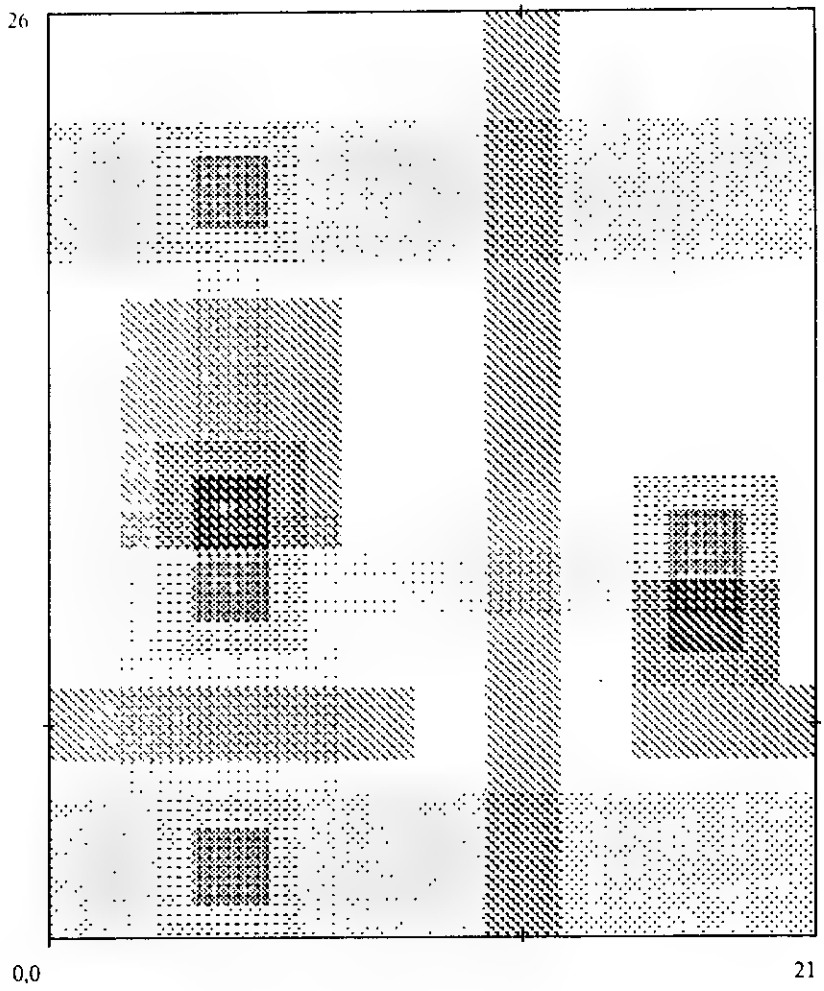


Figure 10a. Check Plot of the SRCELL

[Dimensions in lambda. Implant layer not shown]

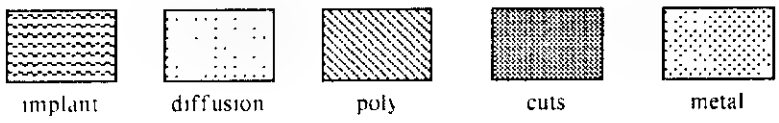


Figure 10b. Check Plot Stipple Codes

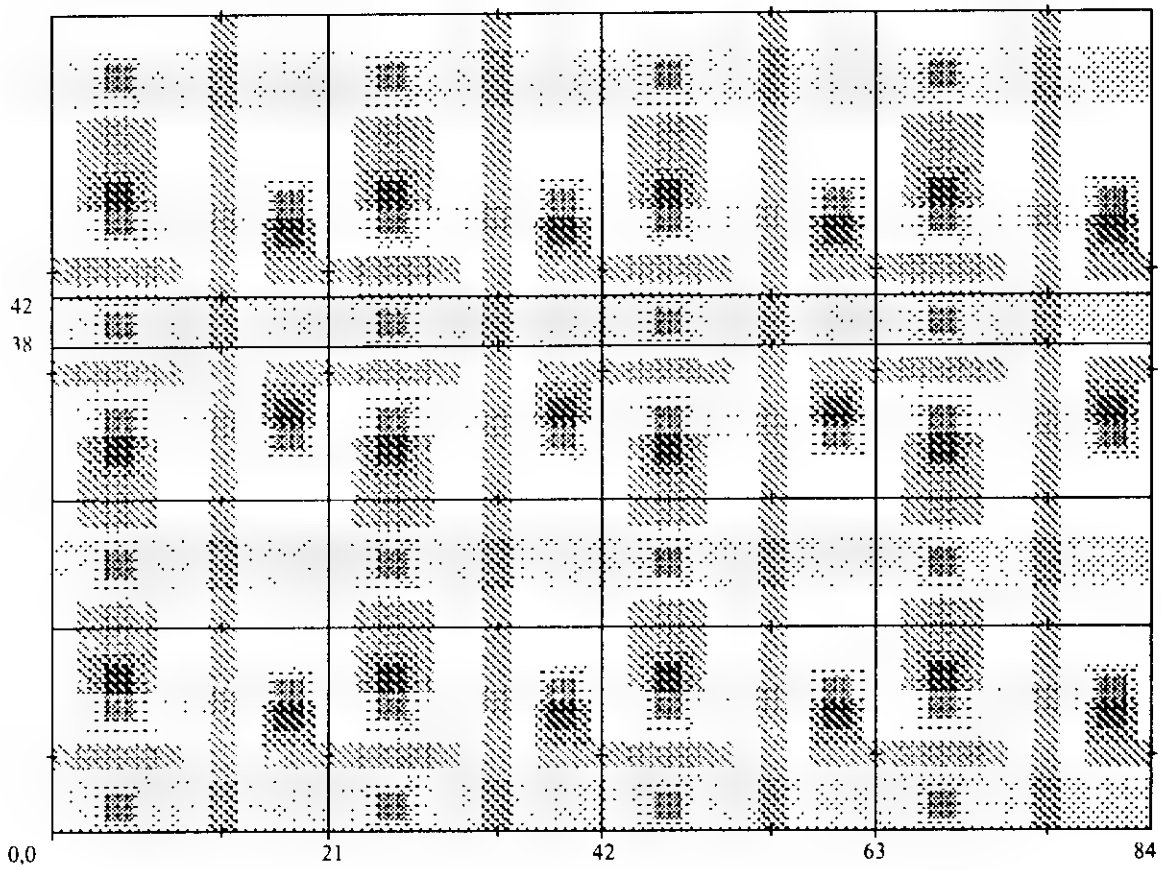


Figure 11. Check Plot of 3 by 4 Array of SRCELLS

[Dimensions in lambda, with the cell outlines indicating relative cell placements according to the program in fig.9]

one SRCELL is given in figure 10a, and we see that the cell has been correctly digitized. A set of stipple patterns is used in this check plot to encode the different system layers, with the coding specified in figure 10b. If available, color checkplots are much better: color checkplots can be made denser and still be readable, and association of colors with layers and functions is more easily made and subject to fewer errors in practice. Note: the implant layer hasn't been plotted in fig. 10a, so that the other layers may be more easily seen.

A check plot of the complete 3 by 4 array of cells is given in figure 11 (again the implant layer is not plotted). Although figure 11 is of insufficient scale to check details within the cells, it enables us to check for correct relative placement of the SRCELLs. The individual cell outlines are included in figure 11, to indicate the nature of the placement of the central row of the array. By mirroring the central row prior to its placement, that row is able to share VDD and GND with the other two rows, thus reducing the overall array size. There is one column of cells per 21 lambda in the x-direction, and one row of cells per 19 lambda in the y-direction. It is *very important* to note that the *outcome of each DRAW statement is determined by the order in which any mirror, rotate, replicate, and translate operations occur* (see the section on the Caltech Intermediate Form, and also R2, Ch.6). Any permutation in the order of these operations may lead to a completely different result.

In chapter 3 we found that the PLA is a useful subsystem structure, often used to implement finite state machines and combinatorial logic. We now present a worked out example of a PLA's layout, to further clarify symbolic layout description. Chapter 3 contains several stick diagrams of PLAs (figs. 13c, 15f). An examination of these stick diagrams reveals that the PLA can be constructed using 6 basic cell types and a slight amount of "random wiring". Once these 6 basic cells have been layed out by hand and symbolically digitized, it is easy to construct symbolic descriptions of different sized PLAs having various numbers of inputs, product-terms, and outputs.

The digitized layouts of four of these basic cells are check plotted in figure 12. The AND and OR planes of the PLA are constructed as arrays of the 14λ by 14λ PLAcellpair cell plotted in figure 12a, which contains two poly and two metal signal lines, and one ground line on the diffusion layer. Diffusion paths may be added in any of four locations in such cells to form transistors, and thus program the PLA. The connection between the AND and OR planes is made using the PLAconnect cell plotted in figure 12b: these cells change the signal paths from the metal to the poly layer. The pullup transistors to be placed at the edges of the AND and OR planes are implemented by the PullupPair cell in figure 12c. The

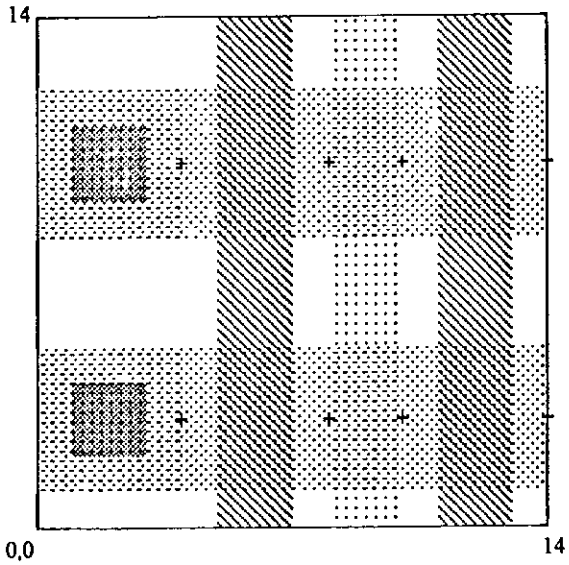


Fig. 12a. PLAcellpair

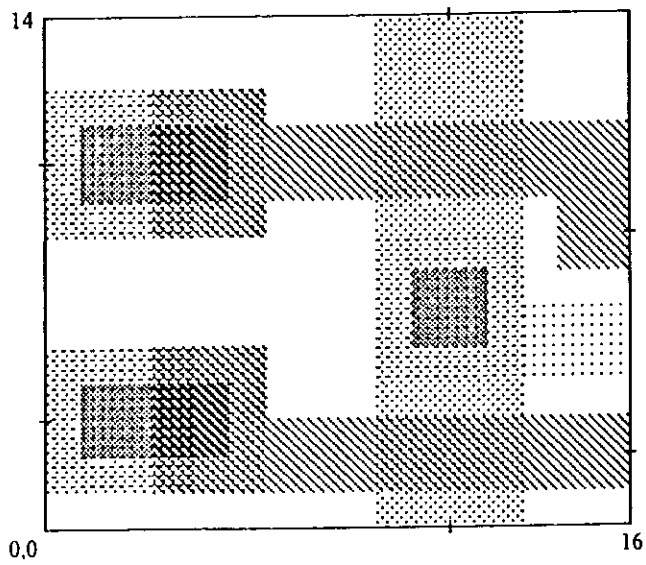


Fig. 12b. PLAconnect

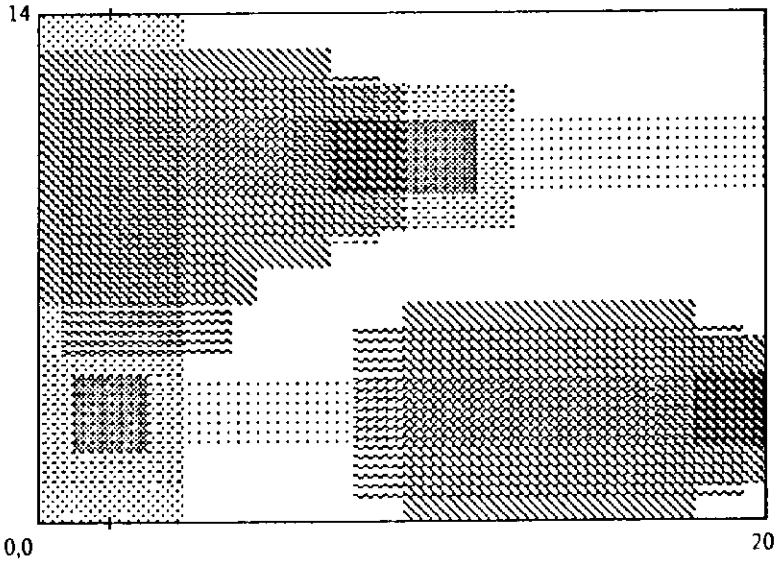


Fig. 12c. PullupPair

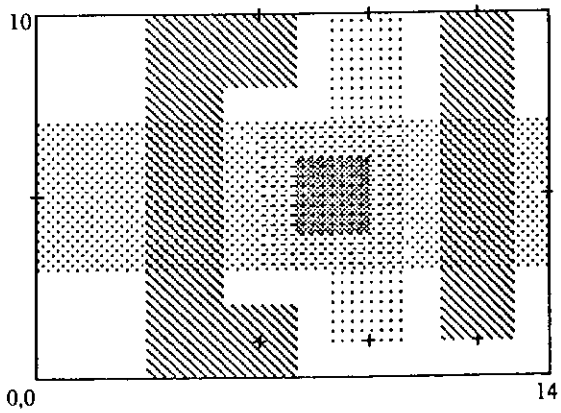


Fig. 12d. PLAground

[all dimensions in lambda]

(placells.press)

ground return paths, to be connected to the diffusion lines crossing the planes, are implemented by the PLAground cell in figure 12d. The PLAground cell is structured so that rows of the cell may be inserted at intervals within AND planes, and columns of the cell inserted at intervals within OR planes, to provide proper ground returns in large PLA's. The two other cell types required are the input drivers and output inverters: these cell layouts are left as exercises for the reader. The cells in figure 12 have been collectively planned so as to fit on a 14λ pitch surrounding the PLA's planes. Figure 13 contains a symbolic description of each of these cell types, and a description of a moderate sized PLA constructed from these cells:

Figure 13. Symbolic Description of a 5-Input, 10-Pterm, 8-Output PLA

```
SCALE    LAMBDA=3.0MICRON;
;
;        PLA CELL DEFINITIONS:
;
SYMBOL   START,PLACELLPAIR;                    [SEE FIGURE 12A.]
  BOX    DIFF,X=0,Y=1,LX=4,LY=4,NY=2,IY=7;
  BOX    DIFF,X=8,Y=0,LX=2,LY=14;              DIFF TO GND
  BOX    POLY,X=5,Y=0,LX=2,LY=14,NX=2,IX=6;
  BOX    CUTS,X=1,Y=2,LX=2,LY=2,NY=2,IY=7;
  BOX    METL,X=0,Y=1,LX=14,LY=4,NY=2,IY=7;    METL TO PULLUPS
SYMBOL   END;
;
SYMBOL   START,PLACONNECT;                     [SEE FIGURE 12B ]
  BOX    DIFF,X=0,Y=1,LX=4,LY=4,NY=2,IY=7;
  BOX    DIFF,X=9,Y=4,LX=4,LY=4;
  BOX    DIFF,X=13,Y=4,LX=3,LY=2;
  BOX    POLY,X=6,Y=1,LX=10,LY=2,NY=2,IY=8;
  BOX    POLY,X=3,Y=1,LX=3,LY=4,NY=2,IY=7;
  BOX    POLY,X=14,Y=7,LX=2,LY=2;
  BOX    CUTS,X=1,Y=2,LX=4,LY=2,NY=2,IY=7;
  BOX    CUTS,X=10,Y=5,LX=2,LY=2;
  BOX    METL,X=9,Y=0,LX=4,LY=14;              GND
  BOX    METL,X=0,Y=1,LX=6,LY=4,NY=2,IY=7,
SYMBOL   END;
;
SYMBOL   START,PULLUPPAIR;                     [SEE FIGURE 12C.]
  BOX    IMPL,X=8.5,Y=0.5,LX=11,LY=5;
  BOX    IMPL,X=0.5,Y=4.5,LX=5,LY=8;
  BOX    IMPL,X=0.5,Y=7.5,LX=9,LY=5;
  BOX    DIFF,X=0,Y=1,LX=4,LY=4;
  BOX    DIFF,X=4,Y=2,LX=16,LY=2;
  BOX    DIFF,X=2,Y=5,LX=2,LY=4;
  BOX    DIFF,X=2,Y=9,LX=18,LY=2;
  BOX    DIFF,X=9,Y=8,LX=4,LY=4;
  BOX    POLY,X=10,Y=0,LX=8,LY=6;
  BOX    POLY,X=18,Y=1,LX=2,LY=4;
  BOX    POLY,X=8,Y=8,LX=2,LY=4;
  BOX    POLY,X=0,Y=7,LX=8,LY=6;
  BOX    POLY,X=0,Y=6,LX=6,LY=1;
  BOX    CUTS,X=1,Y=2,LX=2,LY=2,NX=2,IX=17;
  BOX    CUTS,X=8,Y=9,LX=4,LY=2;
  BOX    METL,X=0,Y=0,LX=4,LY=14;              VDD
  BOX    METL,X=7,Y=8,LX=6,LY=4;
  BOX    METL,X=17,Y=1,LX=4,LY=4;
SYMBOL   END;
```

```
;
SYMBOL   START,PLAGROUND;                          [SEE FIGURE 12D.]
  BOX    DIFF,X=8,Y=1,LX=2,LY=9;
  BOX    DIFF,X=6,Y=3,LX=4,LY=4;
  BOX    POLY,X=3,Y=0,LX=2,LY=10;
  BOX    POLY,X=5,Y=0,LX=2,LY=2NY=2,IY=8;
  BOX    POLY,X=11,Y=1,LX=2,LY=9;
  BOX    CUTS,X=7,Y=4,LX=2,LY=2;
  BOX    METL,X=0,Y=3,LX=14,LY=4;                  GND
SYMBOL   END;
;
SYMBOL   START,PLAINPUT;
;
[ insert symbol definition; size: 14 wide by ~35 high ]
;
SYMBOL   END;
;
SYMBOL   START,PLAOUTPUT;
;
[ insert symbol definition; size: 14 wide by ~41 high ]
;
SYMBOL   END;
;
;          LAYOUT 5-INPUT,10-PTERM,8-OUTPUT PLA:
;                 [SEE FIGURE 14]
;
  DRAW   PLACELLPAIR,NX=5,NY=5,IX=14,IY=14,X=0,Y=0;
  DRAW   PLACONNECT,NY=5,IY=14,X=70,Y=0;
  DRAW   PULLUPPAIR,NY=5,IY=14,X=-19,Y=0;
  DRAW   PLAGROUND,NX=5,NY=2,IX=14,IY=79,X=0,Y=-10;
  DRAW   PLACELLPAIR,ANGLE=270,NX=4,NY=5,IX=14,IY=14,X=86,Y=14;
  DRAW   PULLUPPAIR,ANGLE=270,NX=4,IX=14,X=86,Y=89;
  DRAW   PLAGROUND,ANGLE=270,NY=5,IY=14,X=141,Y=14;
  DRAW   PLAINPUT,NX=5,IX=14,X=0,Y=-44;
  DRAW   PLAOUTPUT,NX=5,IX=14,X=86,Y=-41;
  BOX    DIFF,X=70,Y=-15,LX=4,LY=4;
  BOX    CUTS,X=71,Y=-14,LX=2,LY=2;
  BOX    METL,X=70,Y=-15,LX=4,LY=4;
  BOX    METL,X=-19,Y=70,LX=4,LY=9;                VDD
  BOX    METL,X=-19,Y=79LX=105,LY=4;               VDD
  BOX    METL,X=82,Y=83,LX=4,LY=6,                 VDD
  BOX    METL,X=142,Y=85LX=9,LY=4;                 VDD
  BOX    METL,X=151,Y=-40,LX=4,LY=129;             VDD
  BOX    METL,X=142,Y=-40,LX=9,LY=4;               VDD
  BOX    METL,X=-19,Y=-15,LX=19,LY=4;              VDD
  BOX    METL,X=-19,Y=-11,LX=4,LY=11,              VDD
  BOX    METL,X=70,Y=71,LX=9,LY=4;                 GND
  BOX    METL,X=70,Y=-7,LX=9,LY=4;                 GND
  BOX    METL,X=70,Y=-24,LX=9,LY=4;                GND
  BOX    METL,X=79,Y=-45,LX=4,LY=45;               GND
  BOX    METL,X=83,Y=-21,LX=3,LY=4;                GND
  BOX    METL,X=142,Y=-21,LX=2,LY=4;               GND
  BOX    METL,X=144,Y=-21,LX=4,LY=21;              GND
  BOX    POLY,X=-4,Y=-43,LX=4,LY=2;                PH1
  BOX    POLY,X=142,Y=-7,LX=15,LY=2;               PH2
;
[ insert the PLA "program", using BOXes on the diffusion
        layer to form transistors in the PLAcellpair cells ]
;
[ insert the PLA's input, output, clock, and power connections ]
;
END;
```

A check plot of the PLA described above is given in figure 14. This check plot has been simplified to include only the outlines of the basic cells, plus the additional wiring necessary to complete the PLA. The dimensions and orientations of the cells may be found by comparing these outlines with the cell details in figure 12. Note that in figure 12 some of

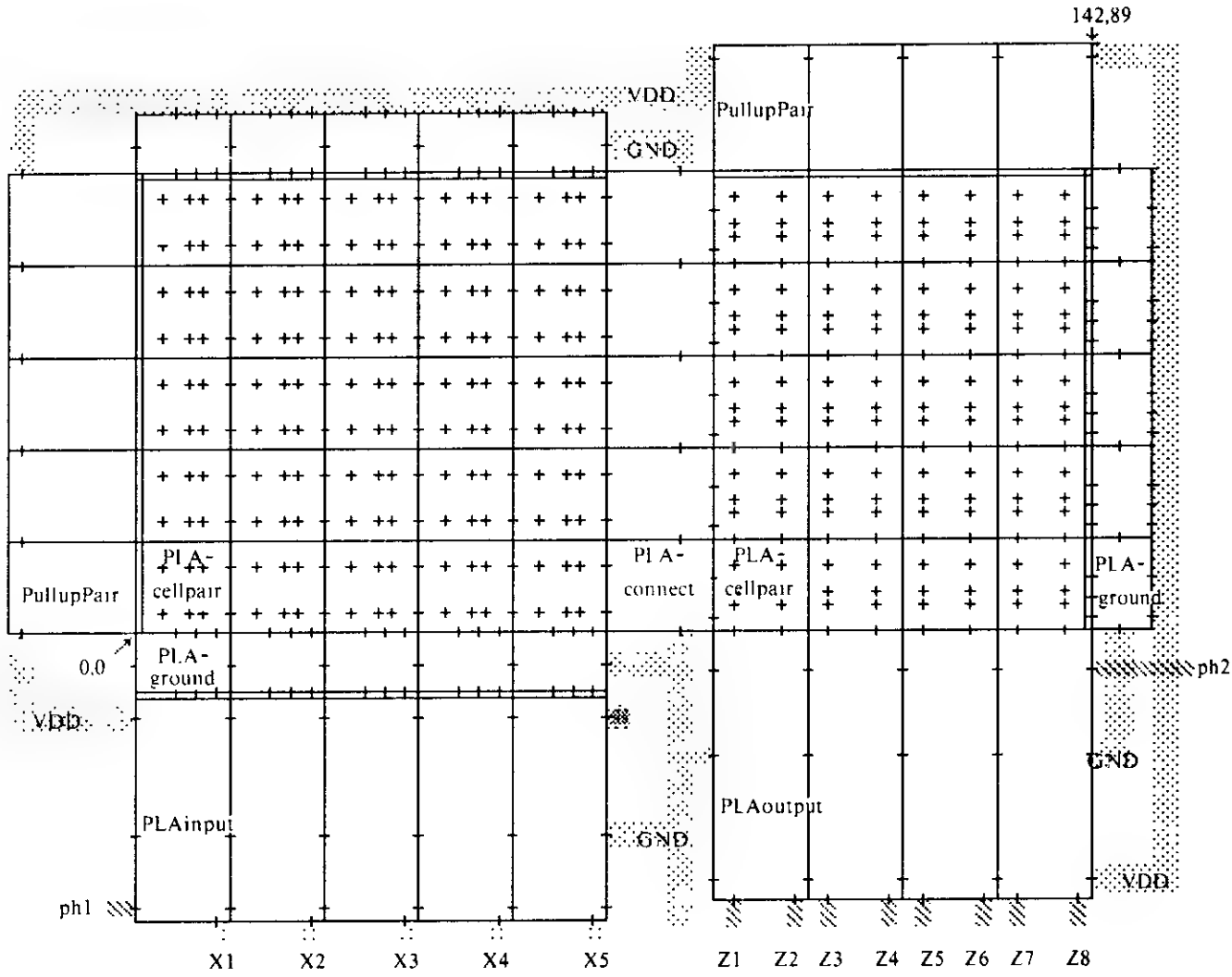


Fig. 14. Check Plot using only Cell Outlines, of the 5-Input, 10-Pterm, 8-Output PLA

[dimensions in lambdas; symbol labels on origin cells of the fig.13 DRAW statements]

the *connection points*, where paths leave or enter at cell edges or where internal connections may be later inserted, are tagged with tick marks. Cell placements and orientations in the check plot may be visualized by locating and identifying the appropriate connection point marks. A comparison of the check plot with the symbolic description above will clarify the function of the various DRAW statements. To assist in this comparison, the origin cell of the array of cells produced by each DRAW statement has been marked in figure 14 with its cell name. Note that this PLA layout could contain the PLA example of chapter 3, fig. 15f.

Symbolic layout languages are easy to define, and may be primitive or sophisticated, according to the requirements of the user. The function of the assembler for such a language is simply to scan and decode the statements and translate them into design files in intermediate form. Conversion of design files into check plot or pattern generator output files is straightforward for the above simple language, since we have used only boxes with a severe constraint on angular orientations. MIRROR and ANGLE transformations are easily handled: x and y coordinates of symbols and boxes are simply replaced by $\pm x$ or $\pm y$, according to the specific parameters, during the instantiation of symbols and drawing of boxes prior to their replication and translation into the layout output file.

The effectiveness of the above language could be further increased by constructing an assembler capable of handling nested sysmbols. By using nested symbols, system layouts may be described in a hierarchical manner, leading to very compact descriptions of structured designs. At the lowest level, one might define symbols for such small but commonly encountered structures as the various forms of contacts. Boxes and these simple symbols could then be used to construct cells such as those in the PLA example above. The PLA could be constructed with these cells, and then defined as a symbol to be used in a larger design. An example of the sort of function one might add to create a much more sophisticated language, and language processor, would be the capability of generating the layout description of a PLA from the collection of basic cells, as a function of its input, pterm, and output size parameters and logic function parameters.

Figure 15 summarizes the procedures and artifacts of hand layout, and layout description and digitization using a layout langauge. By studying figure 15, and thinking back over the material and examples of this section, one can visualize a complete, though primitive, sequence of steps sufficient to prepare a design for implementation. These procedures are entirely adequate for preparing small LSI projects for implementation. The procedures may also be used for those large LSI systems which have highly structured designs.

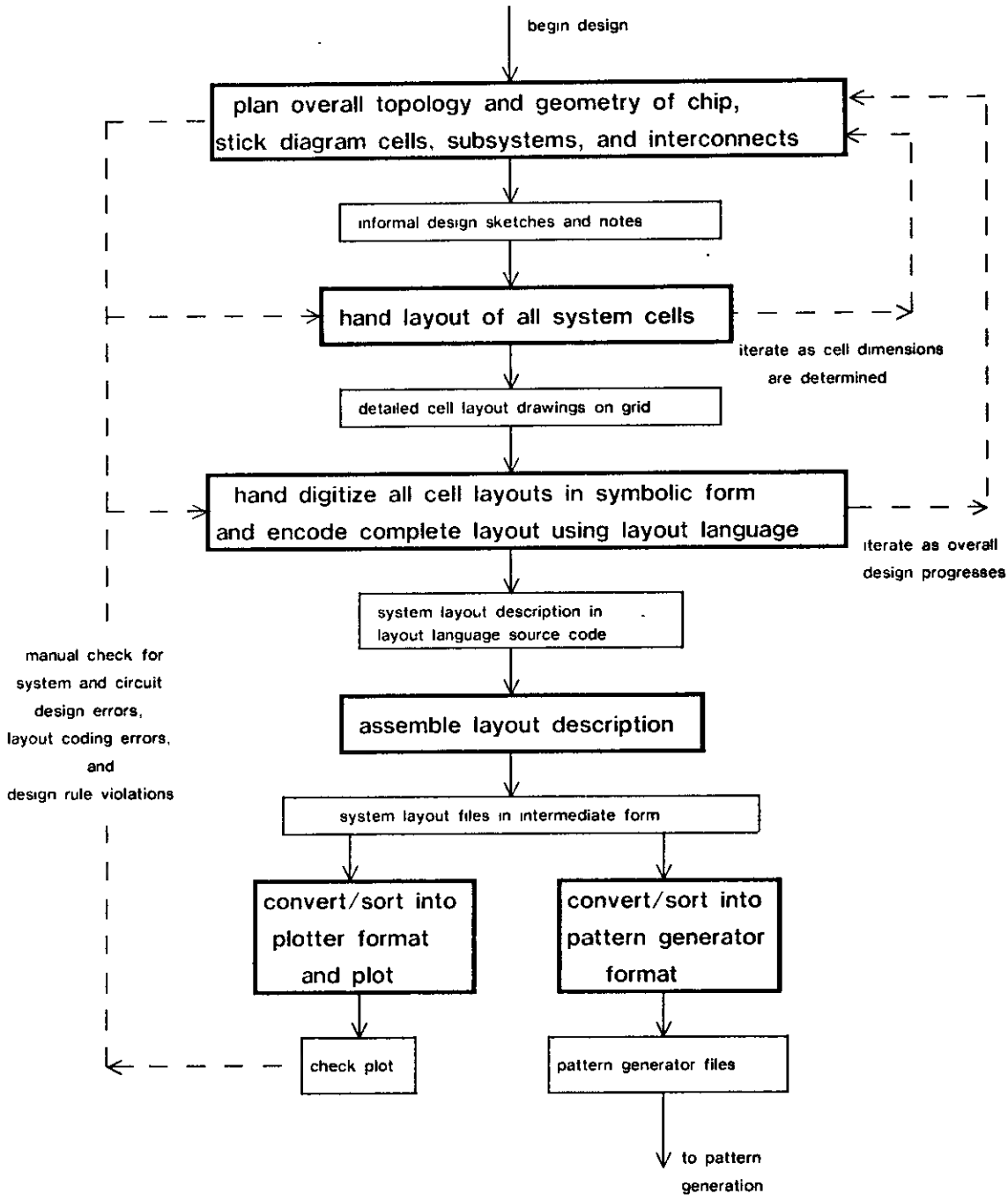


Fig.15. Design, Hand Layout, Design File Generation, And Design Checking Using a Symbolic Layout Language and Simple Machine Aids

The primary obstacle that these primitive procedures place in the path of the system designer is the sheer time and effort it takes to get through the loop to a new check plot each time a small design change is made. The enthusiasm aroused by a sudden insight, such as the conception of a completely new topological possibility for an important system cell, can be dampened by the tedious tasks of hand layout and box digitization required before one can really see the full effect of the idea on the overall system layout.

Though often supported by large batch mode CAD systems for containing, modifying, checkplotting, and simulating designs, the majority of LSI layout now done in industry begins with hand layout. Digitization is usually simplified by the use of digitizing tables, which are much like graphics plotters in reverse: a new section of a design, laid out by hand, is placed on the table and digitized by tapping switches while manually following the outlines of the cell's boxes with a pointer. Although this is less tedious than digitization using a layout language, it is still time-consuming and hardly interactive.

The next section describes an interactive graphics layout system which enables the system designer to quickly sketch new layout ideas and see their effect immediately.

An Interactive Layout System

[Section contributed by Douglas Fairbairn, Xerox PARC, and James Rowson, Caltech]

Computing hardware of sufficient power to support highly interactive graphics has in the past quite expensive, and this has inhibited the widespread application of interactive computing techniques. However, because of expected advances in VLSI technology, we are rapidly approaching the day when many will have access to personal computers with computing power rivaling today's medium to large-scale systems. It will be more difficult to provide effective software for these systems than it will be to build the computers themselves.[3] In this section we describe a highly interactive layout system which runs on a modest personal computer, rather than on an expensive, limited access, centralized system. This system was developed anticipating the work environment of the future, in which most "knowledge" workers will have personal computers as part of their normal office equipment.

ICARUS[1] (Integrated Circuit ARtwork Utility System) is a software system which enables the user to create and modify an integrated system layout directly on a CRT display screen. ICARUS was conceived with the idea that the designer would create and edit a layout at the display, without doing any more than a rough sketch or "stick diagram" before beginning work. Creating and moving items is fast and easy enough so that the designer can truly sketch on the screen. Once the layout is basically correct, the items can be moved or modified to arrive at the most compact layout.

The user is required to remember very little about the available commands or their use because the commands themselves are displayed on the screen and the system prompts the user for additional information as it is needed. The system can format and output check plots to matrix type printers or on raster-scan laser printers. ICARUS design files can be used to create standard pattern generation files from which masks can be made. An overview of design and layout procedures using the system is given in figure 16. It is instructive to compare this with figure 15, which presents equivalent steps for hand layout.

All the software to accomplish these various steps runs on a small experimental minicomputer known as the Alto. This machine was designed by researchers at Xerox PARC as a general purpose personal computer suitable for both text and graphics applications[3]. No additional, special hardware is used by ICARUS. The ICARUS system is programmed in BCPL, an ALGOL-like high level language. There are about 30K words of

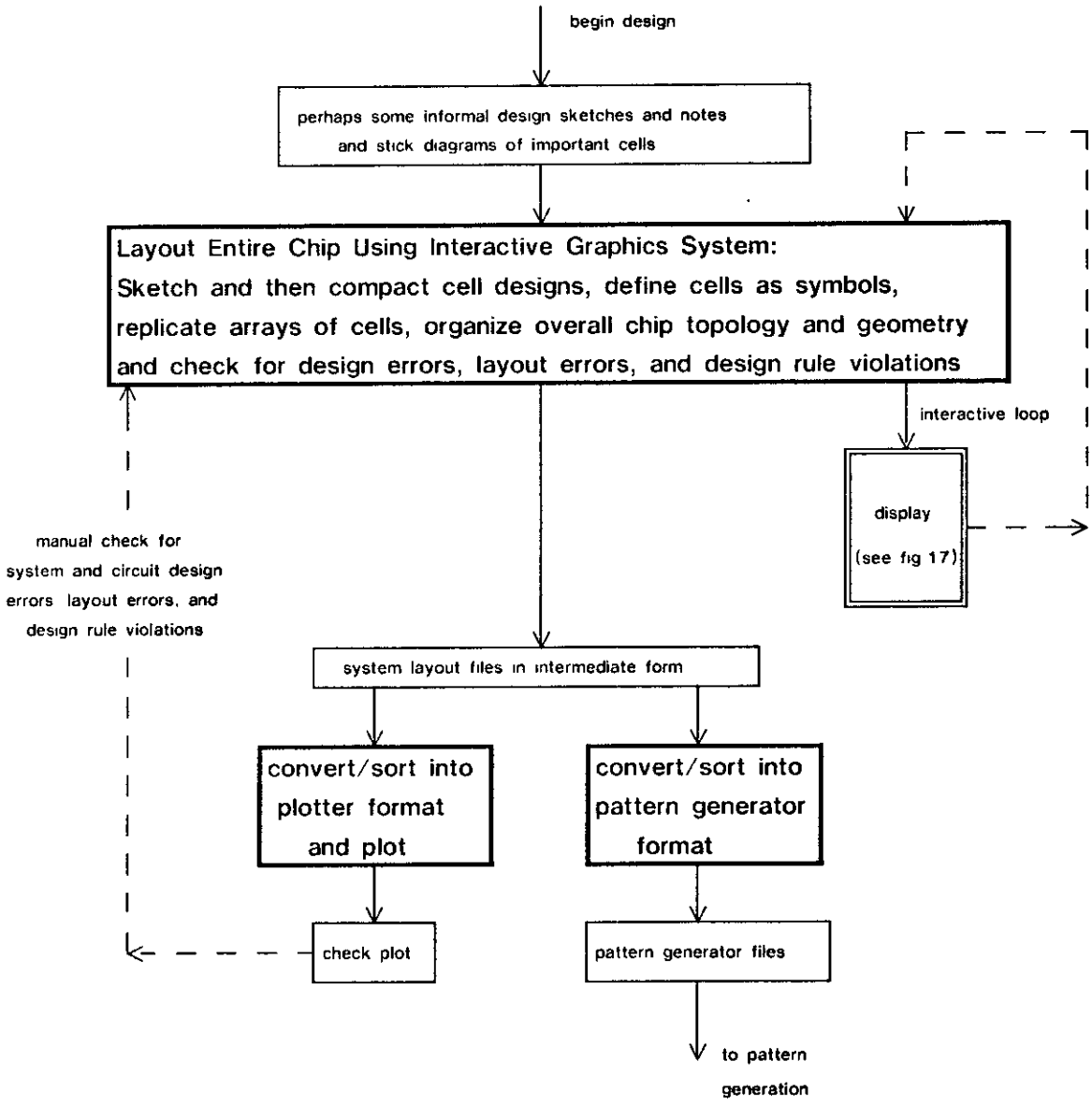


Fig. 16. Design, Layout, Design File Generation, and Design Checking Using an Interactive Graphics Layout System

compiled code in the system of which half is in memory at any given time. At minimum, the Alto memory has 64K 16 bit words. A 2.5 Mbyte cartridge disk drive is an integral part of the system. The user interacts with the system through an unencoded keyboard (software definable keys) and with a pointing device called a mouse (R2, p173). A cursor is controlled on the screen by moving the mouse around on a small area of the user's desk. A bit map display with a resolution of 600x800 dots is used for output, and printers for doing check plots are available through an in-house computer network.

The ICARUS display features two windows which provide a flexible working view of the layout, as shown in figure 17. The upper window is normally used for viewing a large piece of the layout at small magnification, and the lower window used for looking at a smaller section in more detail. The magnifications of the windows may be set independently.

In addition to the two windows there are various menus and status lines presented in the display. The menu on the left is the *command menu.* The menu under the upper window is the *parameter menu.* Under the parameter menu is the *stipple menu*, containing the mask level codes. Rectangles at a given level are stippled with the pattern for that level. The patterns were chosen so that, where necessary, one pattern could be seen through the other to verify that appropriate layers are overlapping properly. Current drawing coordinates and the status of system memory space are displayed to the right of the stipple menu.

The user interface is implemented principally through the display, the mouse and five conveniently located keys on the keyboard. Frequently used commands are given using only one or two simple hand operations, and can be done without glancing away from the display. These characteristics, coupled with rapid display redrawing, enhance the system's interactiveness.

The internal data representation in ICARUS is based on three types of items: rectangles, symbols, and text strings. The organization of these items into memory data structures, and the typical run-time memory space allocation is illustrated in figure 18.

Rectangles are created with the aid of the mouse. They may have angular orientations which are integer multiples of 45°. They can be moved, copied, or deleted using the mouse and one key. As items are created, they are added to an item list in main memory. Each rectangle is stored as 6 words in memory: the first word is the pointer to the next item, the second specifies what layer it is on, what type of item it is, etc. The third through sixth

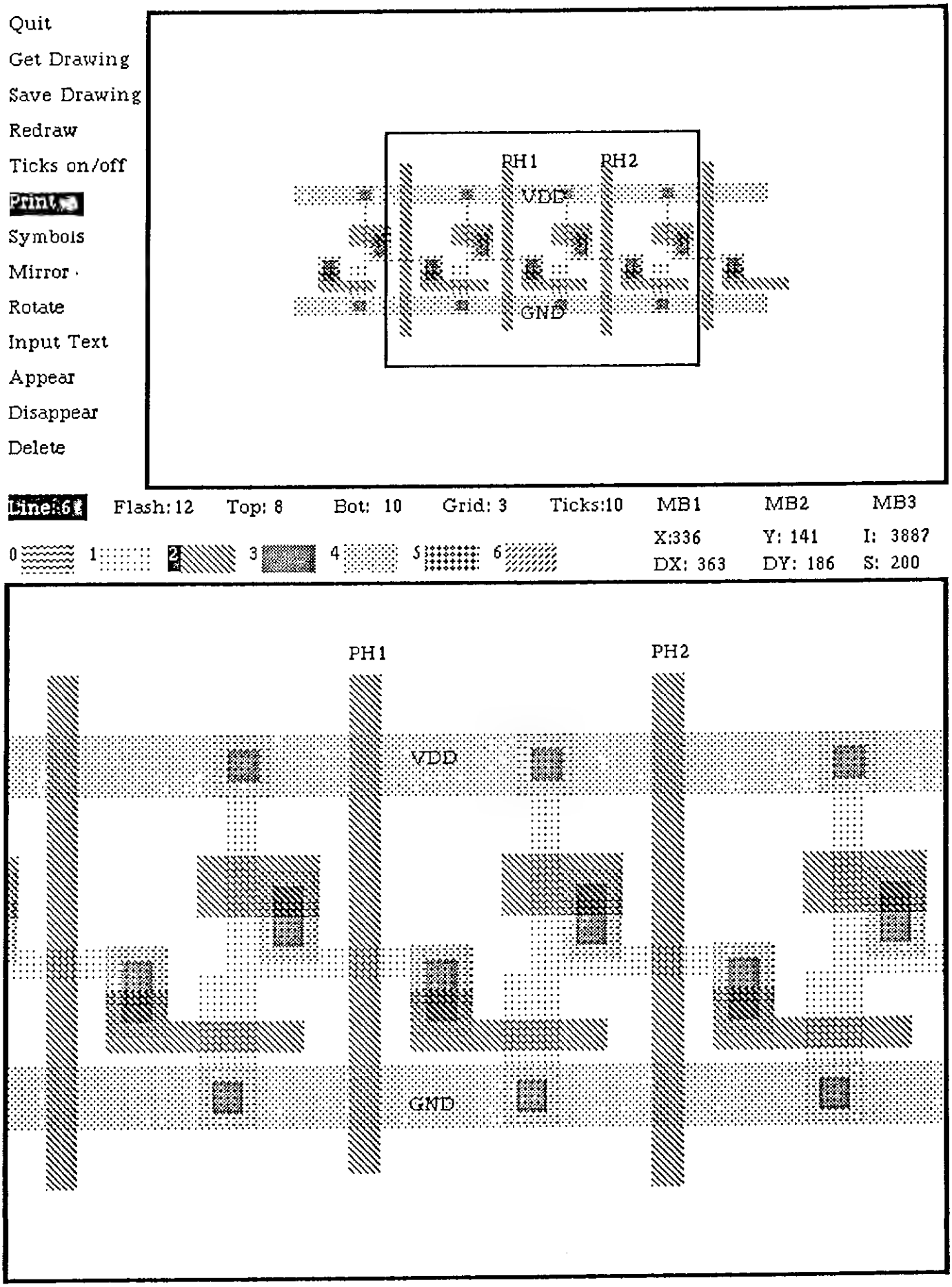


Fig. 17. The ICARUS Display: 2 Views of a Layout in Progress (windows.press)

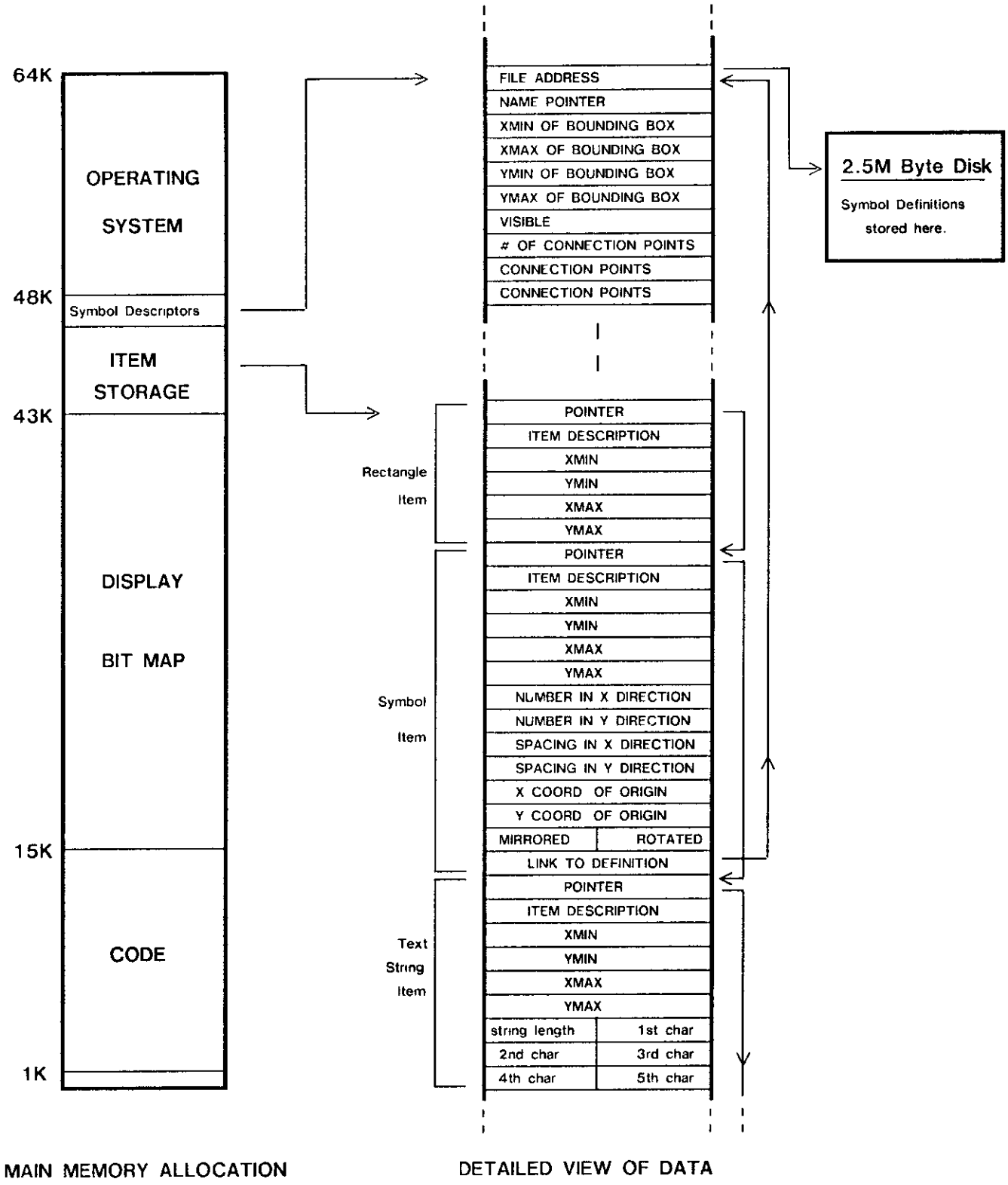


Figure 18. ICARUS Memory Allocation and Data Structure

words specify the minimum and maximum x and y coordinates. The items are kept in order of increasing values of minimum x coordinate, so that the display may be quickly redrawn.

When a symbol is defined by the user, the items which are contained within it are stored on the disk, while a pointer, the name and the bounding box for the symbol are placed in main memory. Symbols can be nested to any level. Once a *symbol definition* has been created, one is free to define *symbol instances* which are references to that definition. The symbol instance may be a command to draw one copy of the symbol at a certain location, or a whole array. The size of the symbol instance, which resides in main memory, is the same in both cases. The use of symbols wherever possible tends to preserve main memory space. Rather large systems can be designed using ICARUS, if the systems are well structured and make extensive use of symbols. This is true even when using a minimum sized 64K memory, which leaves little space for layout data.

Text is used for identifying data and control lines and is merely a memory aid to the user. There is no attempt to make use of the text or other information in the drawing for connectivity or other types of checking.

Operations more complex than those such as draw and move are implemented through the use of menus as shown in figure 17. The desired command is chosen by pointing at it with the cursor and clicking a mouse button. The selected command is then inverted to white on black video to identify its selection, which the user then confirms with a key on the keyboard. At this point, the system prompts the user with instructions presented in the display area normally holding the stipple menu. The instructions lead the user through the individual steps required, for example, to mirror or rotate a group of items.

Operations on symbols are defined in a secondary menu which can be reached by selecting the command "symbols" on the primary menu. The secondary menu offers commands such as define symbol, draw symbol, list the names of the symbols is the symbol library, or expand symbol. This last command is used to modify a symbol which is already defined, the modified symbol definition immediately updating all symbol instances which point to it.

Various system parameters are displayed in the parameter line directly below the top window. Values such as the default line width for the currently selected layer, the magnification of the top and bottom windows, and the spacing of the tick marks are all displayed. The parameter values can be changed at any time by selecting the desired one

and typing the new parameter value on the keyboard. The X,Y layout coordinates of the point last clicked with the mouse are displayed at the right of the screen. The DX,DY distances between the last two clicks are also displayed. This feature provides a convenient "ruler" for measuring distances on the layout.

The construction of an interactive layout system such as ICARUS is a relatively straightforward task for one who is experienced in interactive computer graphics (R2), given a display oriented minicomputer system and effective systems building software. A first version of ICARUS was constructed in 3 man-months, and a mature version produced in an additional 4 man-months.

ICARUS has been used internally in Xerox to lay out many integrated system projects, and to organize a number of multi project chips. Among the users were a number of individuals previously unfamiliar with integrated circuit layout, who nevertheless successfully completed LSI projects with up to 10,000 transistors. We find that the interactive nature of such a system not only aids the experienced designer, but also enhances the learning process for the novice. We believe that such interactive, personal design systems greatly enhance the creative ability of the designer by enabling easy generation and examination of many more design alternatives per unit time than would be the case with centralized, non-interactive design systems.

However, there is more to integrated system design than circuit layout. Design rules must be checked, logic transfer functions tested, and, in certain cases, circuit transfer functions computed to determine delays and predict system performance. We believe that the direction in which to search for further improvements in design tools is in the replacement of the primitive ICARUS type of data structure with one which allows design functions other than just layout to also interactively operate upon the same data base. This is the subject of the later section on fully integrated design systems.

The Caltech Intermediate Form for LSI Layout Description

[Section Contributed by Robert F. Sproull, Carnegie Mellon University, and Richard F. Lyon, Xerox PARC]

The Caltech Intermediate Form (CIF Version 2.0) is a means of describing graphic items (mask features) of interest to LSI circuit and system designers. Its purpose is to serve as a standard machine readable representation from which other forms can be constructed for specific output devices such as plotters, video displays, and pattern-generation machines. The intermediate form is not intended as a symbolic layout language: CIF files will usually be created by computer programs from other representations, such as a symbolic layout language or an interactive design program. Nevertheless, the form is a fairly readable text file, in order to simplify combining files and tracing difficulties.

The basic idea of the form is to specify literally every geometric object in the design using ample precision. Use of this form provides participating design groups easy access to output devices other than their own, enables sharing designs with others, allows combining several designs to form a larger chip, and the like. It is not necessary for all participating groups to implement the entire set of features of CIF, as long as their programs and documents contain warnings about unimplemented functions; nevertheless, the syntax must be correctly interpreted by all programs that read CIF, to assure a reasonable result.

CIF thus serves as the common denominator in the descriptions of various integrated system projects. No matter what the original input methods are (hand layout and coding, or a design system), the designs will be translated to CIF as an intermediate, before being translated again to a variety of formats for output devices or other design aids.

The original CIF was conceived by Ivan Sutherland and Ron Ayers in 1976. Subsequent improvements were contributed by Carlo Sequin, Douglas Fairbairn, and Stephen Trimberger.

This specification is divided into four parts: a description of the syntax of the form, a description of the semantics, an explanation of the transformations used, and a discussion of the conversion of wires to boxes.

Syntax

A CIF file is composed of a sequence of characters in a limited character set. The file contains a list of commands, followed by an end marker; the commands are separated with semicolons. Commands are:

Command	Form
Polygon with a path	P path
Box with length, width, center, and direction (direction defaults to (1,0) if omitted)	B integer integer point point
Round flash with diameter and center	R integer point
Wire with width and path	W integer path
Layer specification	L shortname
Start symbol definition with index, a, b (a and b both default to 1 if omitted)	DS integer integer integer
Finish symbol definition	DF
Delete symbol definitions	DD integer
Call symbol	C integer transformation
User extension	digit userText
Comments with arbitrary text	(commentText)
End marker	E

A more formal definition of the syntax is given below. The standard notation proposed by Niklaus Wirth[14] is used: production rules use equals = to relate identifiers to expressions, vertical bar | for or, and double quotes " " around terminal characters; curly brackets {} indicate repetition any number of times including zero; square brackets [] indicate optional factors (i. e. zero or one repetition); parentheses () are used for grouping; rules are terminated by period. Note that the syntax allows blanks before and after commands, and blanks or other kinds of separators (almost any character) before integers, etc. The syntax reflects the fact that symbol definitions may not nest.

```
cifFile              = { { blank } [ command ] semi }  endCommand { blank }.
command              = primCommand | defDeleteCommand |
                       defStartCommand semi { { blank } [ primCommand ] semi } defFinishCommand.
primCommand          = polygonCommand | boxCommand | roundFlashCommand | wireCommand |
                       layerCommand | callCommand | userExtensionCommand | commentCommand.

polygonCommand       = "P" path.
boxCommand           = "B" integer sep integer sep point [ sep point ].
roundFlashCommand    = "R" integer sep point.
wireCommand          = "W" integer sep path.
layerCommand         = "L" { blank } shortname.
defStartCommand      = "D" { blank } "S" integer [ sep integer sep integer ].
defFinishCommand     = "D" { blank } "F".
defDeleteCommand     = "D" { blank } "D" integer.
callCommand          = "C" integer transformation.
userExtensionCommand = digit userText.
commentCommand       = "(" commentText ")".
endCommand           = "E".
```

```
transformation      = { { blank } ( "T" point | "M" { blank } "X" | "M" { blank } "Y" | "R" point ) }.

path                = point { sep point }.
point               = sInteger sep sInteger.

sInteger            = { sep } [ "-" ] integerD.
integer             = { sep } integerD.
integerD            = digit { digit }.

shortname           = c [ c ] [ c ] [ c ].
c                   = digit | upperChar.
userText            = { userChar }.
commentText         = { commentChar } | commentText "(" commentText ")" commentText.

semi                = { blank } ";" { blank }.
sep                 = upperChar | blank.
digit               = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9".
upperChar           = "A" | "B" | "C" | ... | "Z".
blank               = any ASCII character except digit, upperChar, "-", "(", ")", or ";".
userChar            = any ASCII character except ";".
commentChar         = any ASCII character except "(" or ")".
```

Semantics

The fundamental idea of the intermediate form is to describe unambiguously the geometry of patterns for LSI circuits and systems. Consequently, it is important that all readers and writers of files in this form have exactly the same understanding of how the file is to be interpreted. Many of the decisions in designing the file format were made to avoid ambiguity or small but troublesome errors: floating point numbers are avoided; there are no iterative constructs, though there may be in future additions to CIF.

A simple file format might include only primitive geometric constructs, such as polygons, boxes, flashes and wires. Unfortunately, the geometric description of a chip with hundreds of thousands of rectangles on it would require an immense file of this sort. Consequently, we have made provision for defining and calling symbols; this should reduce the size of the file substantially.

It is important that programs processing CIF files operate cautiously, maintaining a constant vigilance for mistakes or entries that will not be processed properly. The description below mentions implementation suggestions or cause for caution inside brackets [].

Measurements. The intermediate form uses a right-handed coordinate system shown in Figure 19a, with x increasing to the right and y increasing upward. (Directions and distances are always interpreted in terms of the front surface of the finished chip, not in terms of the various sizes and mirrorings of the intermediate artifacts.) The units of distance measurement are hundredths of a micron (μm); there is no limit on the size of a number. [Programs reading numbers from CIF files should check carefully to be sure that the number does not overflow the number of bits in the internal representation used, and should specify their own limits, if any.]

Directions. Rather than measure rotation by angles, CIF uses a pair of integers to specify a "direction vector." (This eliminates the need for trigonometric functions in many applications, and avoids the problem of choosing units of angular measure.) The first integer is the component of the direction vector along the x axis; the second integer along the y axis. Thus a direction vector pointing to the right (the +x axis) could be represented as direction (1 0), or equivalently as (17 0); in fact, the first number can be any positive integer as long as the second is zero. A direction vector pointing NorthEast (i.e., rotated 45 degrees counterclockwise from the x axis) would have direction (1 1), or equivalently (3 3), and so on. [A (0 0) direction vector may be defaulted to mean the +x axis; a warning should be generated].

Geometric primitives. The various primitives that specify geometric objects are not intended to be mutually exclusive or exhaustive. CIF may be extended occasionally to accommodate more exotic geometries. At the same time, it is not necessary to use a primitive just because it is provided. Notice in the examples below that lower case comments and other characters within a command are treated as blanks, and that blanks and upper case characters are acceptable separators.

Boxes: Box Width 60 Length 25 Center 80,40 Direction -20,20; (or B60 25 80 40 -20 20;)
The fields which define a box are shown graphically in Figure 19a. Center and direction (optional, defaults to +x axis) specify the position and orientation of the box, respectively. Length is the dimension of the box parellel to the direction, and Width is the dimension perpendicular to the direction.

Polygons: Polygon A 0,0 B 10,20 C -30,40; (or P0 0 10 20 -30 40;)
A polygon is an enclosed region determined by the vertices given in the path, in order. For a polygon with n sides, n vertices are specified in the path (the edge connecting the last vertex with the first is implied; see Figure 19b). [Programs that try to interpret polygons may place

various restrictions on their paths; no set of constraints has been generally accepted, and no program currently exists for converting completely general polygons to pattern generator output.]

Flashes: RoundFlash Diam 200 Center -500,800; (or R200 -500 800;)
The diameter of a flash is sufficient to specify its shape, and the center specifies its position. (see Figure 19b). [Some programs may substitute octagons, or other approximations, for round flashes.]

Wires: Wire Width 50 A 0,0 B 10,20 C -30,40; (or W50 0 0 10 20 -30 40;)
It is sometimes convenient to describe a long, uniform width run by the path along its centerline. We call this construct a wire (see Figure 19b). An ideal wire is the locus of points within one half-width of the given path. Each segment of the ideal wire therefore includes semicircular caps on both ends. Connecting segments of the wire is a transparent operation, as is connecting new wires to an existing one: the semicircular overlap ensures a smooth connection between segments in a wire and between touching wires. [For output devices that have a hard time constructing circles, we approximate the ideal wire with squared-off ends. Notice that squared-off ends work nicely for segments meeting at right angles, but cause problems if wires or wire segments are connected at arbitrary angles. A way to circumvent this problem is to convert, prior to output, any wires in a file into connected sets of boxes of appropriate length, width, angle and center position (Figure 19c). The width of each box is the same as the width of the wire. The length of the boxes must be adjusted to minimize unfilled wedges and overlapping "ears". An algorithm for constructing boxes from a wire description is given in a later subsection. If the wire is specified within a symbol definition, the approximation need be computed only once, and can then be used each time the symbol is instantiated.]

Layer specification: Layer ND nmos diffusion; (or LND;)
Each primitive geometry element (polygon, box, flash, or wire) must be labeled with the exact name of a fabrication mask on which it belongs. Rather than cite the name of the layer for each primitive separately, the layer is specified as a "mode" that applies to all subsequent primitives, until the layer is set again (layer mode is preserved across symbol calls, which are discussed later).

The argument to the layer specification is a short name of the layer. Names are used to improve the legibility of the file, and to avoid interfering with the various biases of designers and fabricators about numbers (one person's "first layer" is another's "last"). [The intention of the layer specification command is to label locally the layer for a particular geometry. It is therefore senseless to specify a box, wire, polygon or flash if no layer has been specified. In order to detect this error, the command LZZZZ is implicitly inserted at the beginning of the file, and as the first command of

a symbol definition (DS; see below). Any attempt to generate geometric output on layer ZZZZ will result in an error.]

It is important that layer names be unique, so that combining several files in intermediate form will not generate conflicts. The general idea is that the first character of the name denotes the technology, and the remainder is mnemonic for the layer. At present, the following layers are defined:

ND	NMOS Diffusion
NP	NMOS Polysilicon
NC	NMOS Contact cut
NM	NMOS Metal
NI	NMOS depletion mode Implant
NB	NMOS Buried contact
NG	NMOS overGlass openings

New layer names will be defined as needed.

[Programs that read CIF will want to check to be sure that layer names used do in fact correspond to fabrication masks being constructed. However, the file may cite layer names not used in a particular pass over the CIF file. It would be helpful for the program to provide a list of the layer names that it ignored.]

Symbols. Because many LSI layouts include items that are often repeated, it is helpful to define often-used items as "symbols." This facility, together with the ability to "call" for an instance of the symbol to be generated at a specific position, greatly reduces the bulk of the intermediate form.

The symbol facilities are deliberately limited, in order to avoid mushrooming difficulties of implementing programs that process CIF files. For example, symbols have no parameters; calling a symbol does not allow the symbol geometry to be scaled up or down; there are no direct facilities for iteration. The main reason for symbol facilities is to limit the file size; if the symbol mechanism is not adequate for some application, the desired geometry can still be achieved with less use of symbols, and more use of explicit geometrical primitives. [Symbols need not be used at all; this eliminates the need for intermediate storage for symbol definitions, but results in larger design files. Machines which must process a fully-instantiated representation of a layer (such as pattern generators) might only accept CIF files without symbol definitions, to reduce the cost of implementation. Therefore, it would be useful to have a program that would convert general CIF files to fully instantiated CIF files, and maybe to sort by layer, location, or whatever.]

The ability to call for iterations (arrays) of symbols is not provided in CIF Version 2.0.

This is primarily due to the difficulty of defining a standard method of specifying iterations, without introducing machine-dependent computation problems. It is still possible to acheive a great deal of file compaction by defining several layers of symbols (e.g. cell, row, double-row, array, etc.). However, the ability to iterate symbol calls is a likely prospect for a future addition to CIF.

Defining symbols: Definition Start #57 A/B = 100/1; . ; Definition Finish; (or DS57 100 1; ... ;DF;)
A symbol is defined by preceding the symbol geometry with the DS command, and following it with the DF command. The first argument of the DS command is an identifying symbol number (unrelated to the order of listing of symbol definitions in the file).

The mechanism for symbol definition includes a convenient way to scale distance measurements. The second and third arguments to the DS command are called a and b respectively. As the intermediate form is read, each distance (position or size) measurement cited in the various commands (polygons, boxes, flashes, wires and calls) in the symbol definition is scaled to (a*distance)/b. For example, if the designer uses a grid of 1 micron, the symbol definition might cite all distances in microns, and specify a=100, b=1. Or the designer might choose lambda (characteristic fabrication dimension) as a convenient unit. This mechanism reduces the number of characters in the file by shrinking the integers that specify dimensions and may improve the legibility of the file (it does not provide scaling, or the ability to change the size of a symbol called within the definition).

Definitions may not nest. That is, after a DS command is specified, the terminating DF must come before the next DS. The definition may, however, contain calls to other symbols, which may in turn call other symbols.

There is only one restriction on the placement of symbol definitions in the file: a symbol must be defined before its instantiation becomes necessary. This constraint can be satisfied by placing all symbol definitions first in the file, and then calls on the symbols. In fact, it is often convenient to have the file consist exclusively of symbol definitions and ONE call on a symbol. This call will be the last command in the file before the end command. [If a definition redefines a symbol that already exists, the previous definition is discarded; a warning message should be generated When several people contribute to a design, some symbol management is therefore necessary; see *Deleting symbol definitions* below.]

Calling symbols: Call Symbol #57 Mirrored in X Rotated to -1,1 then Translated to 10,20;
The C command is used to call a specified symbol and to specify a transformation that should be applied to all the geometry contained in the symbol definition. The call command identifies the symbol to be called with its "symbol index," established when the symbol was defined.

The transformation to be applied to the symbol is specified by a list of primitive transformations given in the call command. The primitive transformations are:

T point	Translate the current symbol origin to this point.
M X	Mirror in X, i.e. multiply X coordinate by -1.
M Y	Mirror in Y, i.e., multiply Y coordinate by -1.
R point	Rotate symbol's x axis to this direction.

Intuitively, each coordinate given in the symbol is transformed according to the first primitive tranformation in the call command, then according to the second, etc. Thus "C1 T500 0 MX" will first add 500 to each x coordinate from symbol 1, then multiply the x coordinate by -1. However, "C1 MX T500 0" will first multiply the x coordinate by -1, and then add 500 to it: the order of application of the transformations is therefore important. In order to implement the transformations, it is not necessary to perform each primitive operation separately; the several operations can be combined into one matrix multiplication (see the subsection on transformations).

Symbol calls may nest; that is, a symbol definition may contain a call to another symbol. When calls nest, it is necessary to "concatenate" the effects of the transformations specified in the various calls (see the subsection on transformations). [There is no sensible way in which a symbol may be invoked recursively (i.e., call itself, either directly or indirectly). Programs that read the intermediate form should check that no recursion occurs. This can be achieved by retaining a single flag with each symbol to indicate whether the symbol is currently being instantiated; the flags are initialized to "false." When a symbol is about to be instantiated, we check the flag; if it is "true," we have detected recursion, print an error message and do not perform the call. Otherwise, we mark the flag "true," instantiate the symbol as specified, and mark the flag "false" when the instantiation is complete.]

Layer settings are preserved across symbol calls and definitions. Thus, in the sequence:

```
LNM;
S6 20 0;
C 57 T45 13;
DS 114...;
DF;
LNM;
S3 0 0;
```

the second LNM is not necessary, regardless of the specification of symbols 57 and 114.

Deleting symbol definitions: Delete Definitions greater than or equal to 100, (or DD100;)
The DD command signals the program reading the file that all symbols with indices greater than or equal to the argument to DD can be "forgotten" -- they will not be instantiated again. This feature is included so that several intermediate form files can be appended and processed as one. In such a case, it is essential to delete symbol definitions used in the first part of the file both because the definitions may conflict with definitions made later and because a great deal of storage can usually be saved by discarding the old definitions.

The argument to DD that allows some definitions to be kept and some deleted is intended to be used in conjunction with a standard "library" of definitions that a group may develop. For example, suppose we use symbol indices in the range 0 to 99 for standard symbols (pullup transistors, contacts, etc.) and want to design a chip that has 2 student projects on it. Each project defines symbols with indices 100 or greater. The CIF file will look like:

```
/Definitions of library symbols;
DS 0 100 1;
/ . definition of symbol 0 in library;
DF;
DS 1 100 1;
/ ...definition of symbol 1;
DF;
/ ..remainder of library;

/Begin project 1;
DS100 100 1;
/ . first student's first symbol definition;
DF;
..
DS109 100 1;
/ ...first student's main symbol definition;
DF;
C109 T403 -110,/ call on first student's main symbol;

DD100;/Preserve only symbols 1 to 99;

/Begin project 2;
DS100 100 1;
/ ...second student's first symbol definition;
DF;
...
DS113 100 1;
/ .. second student's main symbol definition;
C1 T-3 45;/Call on library symbol, still available;
DF;
C113 T401 0;/ call on second student's main symbol;

E
```

User expansion: 3'SYMBOL.LIBRARY', 5.NONSTANDARD DESIGN RULES: LAMBDA = 4.0;

Several command formats (any command starting with a digit) are reserved for expansion by individual users; the authors of the intermediate form agree never to use these formats in future expansions of the standard format. For example, private expansions might provide for (1) requesting that another file be "inserted" at this point in the processing, thus simplifying the use of symbol libraries; (2) inserting instructions to a preprocessor that will be ignored by any program reading only standard intermediate form constructs; or (3) recording ancillary information or data structures (e.g., circuit diagrams, design-rule check results) that are to be maintained in parallel with the geometry specified in the style of the intermediate form.

Comments: (HISTORY OF THIS DESIGN:);
The comment facility is provided simply to make the file easier to read. [It is possible to deactivate any number of commands by simply enclosing them within a pair of parentheses, even if they already include balanced parentheses.]

End Command: End of file.
The final E signals the end of the CIF file. [Programs that read CIF should gave an error message if the file ends without an End Command, or a warning if more text other than blanks follows the E.]

Transformations (see also reference R2)

When we are expanding a symbol, we need to apply a transformation to the specification of an item in the symbol definition to get the specification into the coordinate system of the chip. There are three sorts of measurements that must be transformed: distances (for widths, lengths), absolute coordinates (for "points" in all primitives) and directions (for boxes).

Distances are never changed by a symbol call, because we allow no scaling in the call. Thus a distance requires no transformation.

A point (x,y) given in the symbol is transformed to a point (x',y') in the chip coordinate system by a 3x3 transformation matrix T:

$$[x'\ y'\ 1] = [x\ y\ 1]\ T$$

[It is a good idea to check either the last column of T, or the 1 at the end of the transformed vector, even though they never need to be computed.]

T is itself the product of primitive transformations specified in the call: T = T1 T2 T3, where T1 is a primitive transformation matrix obtained from the first transformation primitive given in the call, T2 from the second, and T3 from the third (of course, there may be fewer or more than 3 primitive transformations specified in the call). These matrices are obtained using the following templates for each kind of primitive transformation:

T a b.	Tn =	1	0	0	
		0	1	0	
		a	b	1	
M X.	Tn =	-1	0	0	
		0	1	0	
		0	0	1	
M Y.	Tn =	1	0	0	
		0	-1	0	
		0	0	1	
R a b.	Tn =	a/c	b/c	0	
		-b/c	a/c	0	where c = Sqrt(a^2+b^2)
		0	0	1	

Transformation of direction vectors (x y) is slightly different than the transformation of coordinates. We form the vector [x y 0], and transform it by T into the new vector [x' y'

0]. The transformed direction vector is simply (x' y'). [Note that some output devices may require rotations to be specified by angles, rather than direction vectors. Conversion into this form may be delayed until necessary to generate the output file. Then we calculate the angle as ArcTan(y/x), applying care when x=0.]

Nested calls require that we combine the transformations already in effect with those specified in the new call. Suppose we are expanding a symbol a, as described above, transforming each coordinate in the symbol to a coordinate on the chip by applying matrix Tac. Now we encounter, in a's definition, a call to b. What is to happen to coordinates specified in b? Clearly, the transformations specified in the call will yield a matrix Tba that will transform coordinates specified in symbol b to the coordinate system used in symbol a. Now these must be transformed by Tac to convert from the system of symbol a to that of the chip. Thus, the full transformation becomes

[x' y' 1] = [x y 1] Tba Tac

The two matrices may be multiplied together to form one transformation Tbc = (Tba Tac) that can be applied to convert directly from the coordinates in symbol b to the chip. This procedure can be carried to an arbitrary depth of nesting.

To implement transformations, we proceed as follows: we maintain a "current transformation matrix" T, which is initialized to the identity matrix. We use this matrix to transform all coordinates. When we encounter a symbol call, we:

1. "Push" the current transformation and layer name on a stack.
2. Set layer name to ZZZZ.
3. Collect the individual primitive transformations specified in the call into the matrices T1, T2, T3 etc.
4. Replace the current transformation T with T1 T2 T4 ... T; i.e., premultiply the existing transformation by the new primitive transformations, in order).
5. Now process the symbol, using the new T matrix.
6. When we have completed the symbol expansion, "pop" the saved matrix and layer name from the stack. This restores the transformation to its state immediately before the call.

Decomposing Wires Into Boxes

The following algorithm for decomposing wires into boxes was developed by Carver Mead, and first implemented at Caltech by Ron Ayers; it was was further modified to be consistent with the use of direction vectors, to allow more general path lengths, and to avoid use of trigonometric functions. [Note that this decomposition covers more area than the locus of points within w/2 of the path for small angles of bend, but less area for sufficiently sharp bends; in particular, if a path bends by 180 degrees (reverses) it will have no extension past the point of reversal (it is missing a full semicircle). Other decompositions are possible, and may better approximate the correct shape.]

Let the wire consist of a path of n points $p_1,...,p_n$.
Let w represent the width of the wire.

```
"Initialization:"
IF n = 0 THEN DONE; "no path"
IF n = 1 THEN
  {MAKEFLASK[Diameter ← w, Center ← p1]; "single-point gets a flash";
  DONE;};
i ← 1;
OldEntension ← w/2; "initial end of wire"
Segment ← p2 - p1; "p1 and p2 are points in path, Segment is a vector (a point)"
"LoopConditions:"
FOR p1, p1+1 in path UNTIL p1+1 is last DO
"calculate the box for the segment from p1 to p1+1:"
IF p1+1 is last THEN { Entension ← w/2; "final end of wire" }
  ELSE
  { "compute Entension for intermediate point:"
  NextSegment ← p1+2 - p1+1; "next vector in path"
  T ← MATRIX[  X[Segment], -Y[Segment],
               Y[Segment],  X[Segment] ];
  "T transforms Segment to +x axis."
  Bend ← MULTIPLY[ NextSegment, T ]; "relative direction vector"
  "if Bend is (0 0), delete p1+1, reduce n, and start over"
  Entension ← w/2 * ( ABS[Y[Bend]] /
              ( LENGTH[Bend] + ABS[X[Bend]] ) );
  };
MAKEBOX [ { Length    ← LENGTH[Segment] + Entension + OldEntension; },
          { Width     ← w; },
          { Center    ← (pi + pi+1)/2 + ( Segment / LENGTH[Segment] ) *
                        (Entension - OldEntension)/2; },
          { Direction ← Segment; "careful, may be zero vector" } ];
i ← i + 1;
OldEntension ← Entension;
Segment ← NextSegment; "next vector in path"
ENDLOOP;
DONE;
```

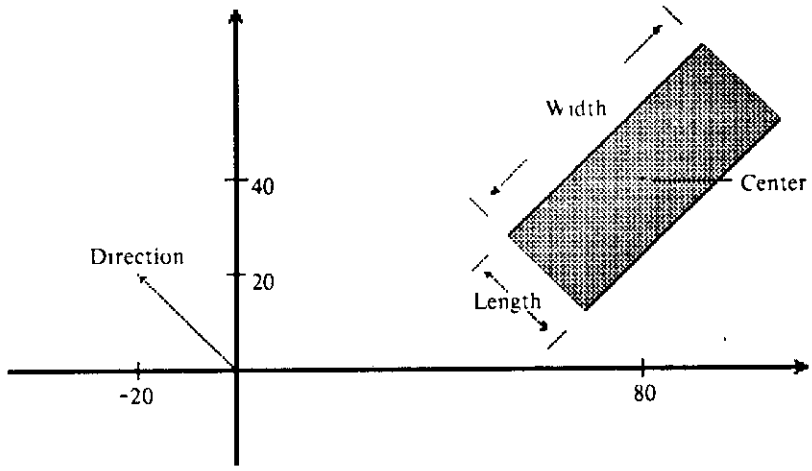


Fig. 19a. Box Representation in Intermediate Form

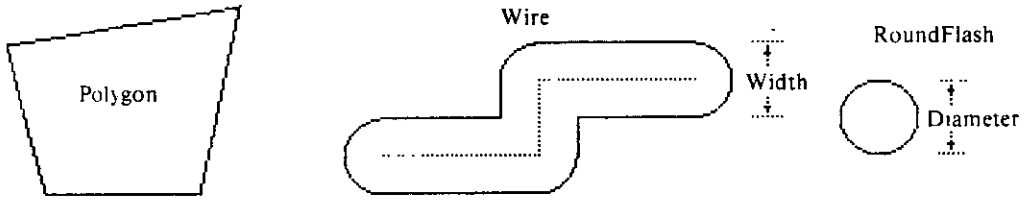
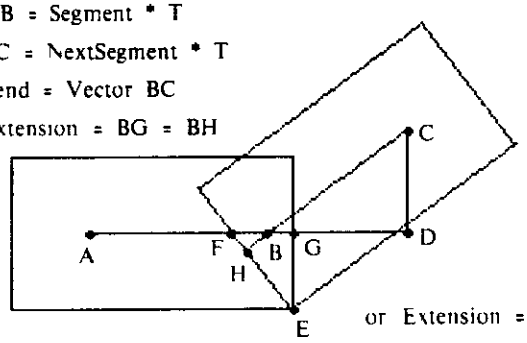


Fig. 19b. Other Items in the Intermediate Form

T transforms Segment to the +X axis
AB = Segment * T
BC = NextSegment * T
Bend = Vector BC
Extension = BG = BH



Similar triangles BCD, EFG, BFH
BC:CD:DB :: EF:FG:GE :: BF:FH:HB
FG = FB + BG
= BH * (BC/DB) + BG
= (1 + BC/DB) * BG
BG = FG / (1 + BC/DB)
= GE * (BC/DB) / (1 + BC/DB)
= GE * (CD / (DB + BC))

or Extension = w/2 * Y[Bend] / (LENGTH[Bend] + X[Bend])

fig. 19c. Converting Wires to Boxes

(cif19.press)

The Multi-Project Chip

Insight into integrated system design is most quickly gained by actually carrying through to completion several LSI design projects, each of increasing scope. A large, complex VLSI system could be quickly and successfully developed by designers able to easily implement and test prototypes of its subsystems. The separate subsystems can be implemented, tested, debugged, and then merged together to produce the overall system layout. However, such activities are only practical if a scheme exists for carrying out implementation with minimum turnaround time and low procedural overhead per project.

In this section we describe procedures for organizing and implementating many small projects by merging their layouts onto one *multi-project chip*. Then each designer of a small project or subsystem need not carry the entire procedural burden involved in maskmaking and fabrication. We also include a collection of practical tips and hints that may prove useful to those undertaking their first projects or organizing their first multi project chips. While the details in this section are specific to present maskmaking and fabrication technology, they nevertheless give a feeling for the sort of things that must be done to implement projects in general. In a later section we discuss how multiple project implementation might be done in the future.

Figure 20 contains a photomicrograph of a Caltech class project chip containing 15 separate student projects. The individual projects were simply merged together onto one typically sized chip layout, approximately 3 mm by 4 mm, and implemented simultaneously as one chip type. Most of these projects are prototypes of digital subsystems designed using the methodology of this text. By implementing a small "slice" of a prototype subsystem array, one can verify that its design, layout, and implementation are correct, and measure its power and delay characteristics as yielded by the particular fabrication process, thus gaining almost as much information as would be obtained by implementing the full array.

Following fabrication, the wafers containing such multi project chips are scribed, diced, and then divided up among the participants. The typical minimum fabrication run is about 10 wafers, each ~7.5 to 10 cm in diameter. Thus even a minimum run provides a few thousand chips, and each participant ends up with many chips. Participants may then each package

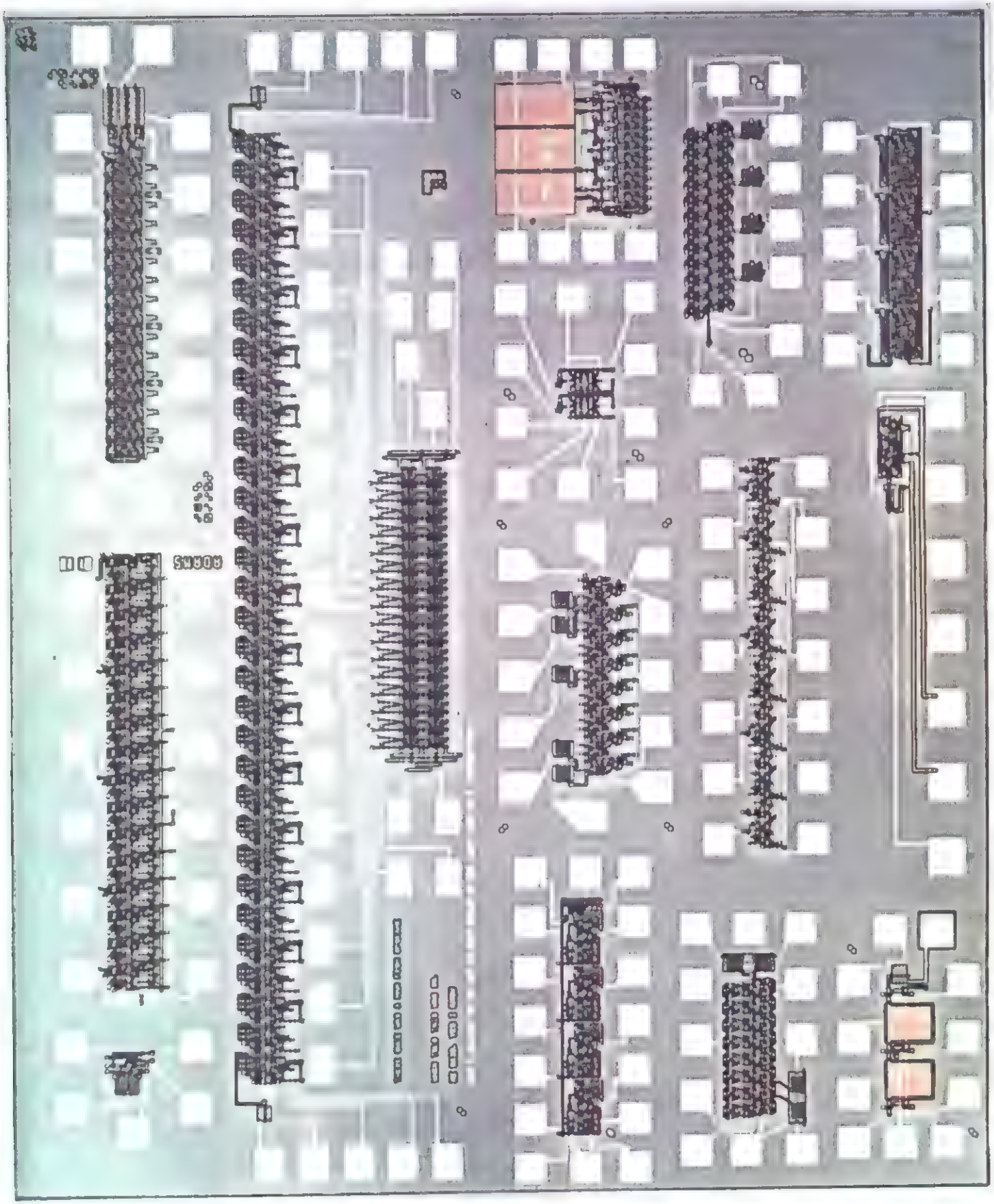


Fig. 20. Photomicrograph of a Caltech Class Project Chip

their chips, bonding the package leads to the contact pads of their individual project. Since most such projects are relatively small in area, yields are unusually high: if a project's design and layout have been done correctly, most of the corresponding chips will work.

Organizing a multi-project chip involves: (i) creating the layout of a *starting frame*, into which the various projects are to be merged, (ii) gathering, relocating, and merging the project layouts into the starting frame to create one design file and generating from this the PG files for the overall project chip, and (iii) documenting various parameters and specifications to be used during maskmaking and fabrication.

The starting frame contains all the auxiliary portions of the chip layout: scribe lines, alignment marks, line width testers (critical dimension marks), and test patterns. The starting frame may contain fiducial marks on each mask level if these are not to be placed by the mask house, and in some cases may contain a parity mark on each level to mark the appropriate reticle side and orientation during step and repeat reduction. A tip: placing a mask level name or symbol somewhere within the chip's scribe line boundary on each level helps prevent the fatal error of level interchange at some time during project merging, maskmaking, or fabrication.

The contents of this starting frame must be carefully worked out to meet the requirements and constraints of the chosen mask house and fab line. The important factor of turnaround time for the entire mask and fab sequence may be reduced to some extent by repeatedly using a relatively standard starting frame which then becomes familiar to all those involved. Some typical values for the time involved: 2 to 3 weeks for maskmaking, and then 3 to 4 weeks for fabrication, longer if large work queues exist at the mask or fab firms.

When a multi-project chip is scheduled, a tentative chip partition for each project can be negotiated among the participants. Project design and layout can then proceed, with iterations on the space allocation being done right up till the final merging. The gathering and merging of project layout files into one design file is simplified if they are in a *common intermediate form*. Projects may then be relocated to their respective partitions of the chip, displayed, plotted, or otherwise checked, using minimum and consistent software operating upon manageable sized files. When the project chip appears correctly organized, pattern generator (PG) files are produced and written on a mag tape to be sent to the mask house.

An alternative to the merging of projects at the intermediate form level, is the relocation

and merging of their PG files. However, the PG files for major designs, containing fully instantiated artwork, become unwieldy in size even at today's complexity. The PG file merging scheme is workable for projects of small to moderate size, and does provide a contingency plan for including projects having alien intermediate forms. If designs are relocated and merged at the PG level, additional software should be provided for displaying or plotting the chip at that level, so that merging errors may be spotted. A tip: it is a good idea in any case to have some bounds checking to prevent stray items of one project from clobbering another.

A thought: the interface between design groups and mask houses would be cleaner if design files in a *common intermediate form*, such as CIF, rather than PG files were used to transmit designs to the patterning process. Files would be much smaller. The use of data links would be eased. The process to convert and sort design files into PG files, involving patterning mechanism dependent optimization, would be appropriately located: in association with the particular patterning mechanism.

Examples of Multi-Project Chips:

The above concepts and some further possibilities may be clarified by examining the details of some specific examples. Figure 21 illustrates a collaborative Xerox PARC/Caltech multi-project chip set [organized by D Fairbairn, D. Johannsen, R. Lyon, J Rowson, S. Trimberger]. The figure was produced as a software blowback from the PG file, of the metal level of this chip set. Projects in the set ranged in scope from the test of a few cells of an experimental, low power shift register [C. Sequin, U C. Berkeley, and R Lyon, Xerox PARC], up to a complete content addressible cache memory system [D. Fairbairn].

Although several of the projects in the set are fairly large, all were individually designed to yield chip sizes packagable in standard 40 pin packages, which can hold chips up to ~ 7 mm square. The pattern generator at the intended mask house was a GCA/D.W.Mann 3600, and the photorepeater was a Mann 3696. Together, these can produce 10x reticles having field sizes as large as 10 cm square, and can reduce, step, and repeat these at a maximum of 10mm x,y intervals onto masks. Therefore, the 3600/3696 can provide masks for square chips up to 10 mm (10,000μ) on a side. A 10mm square chip can hold the patterns of several normally sized chips. By including *interior scribe lines* in the starting frame, as indicated in figure 21, one reticle set can be patterned on the Mann 3600 to contain a number of

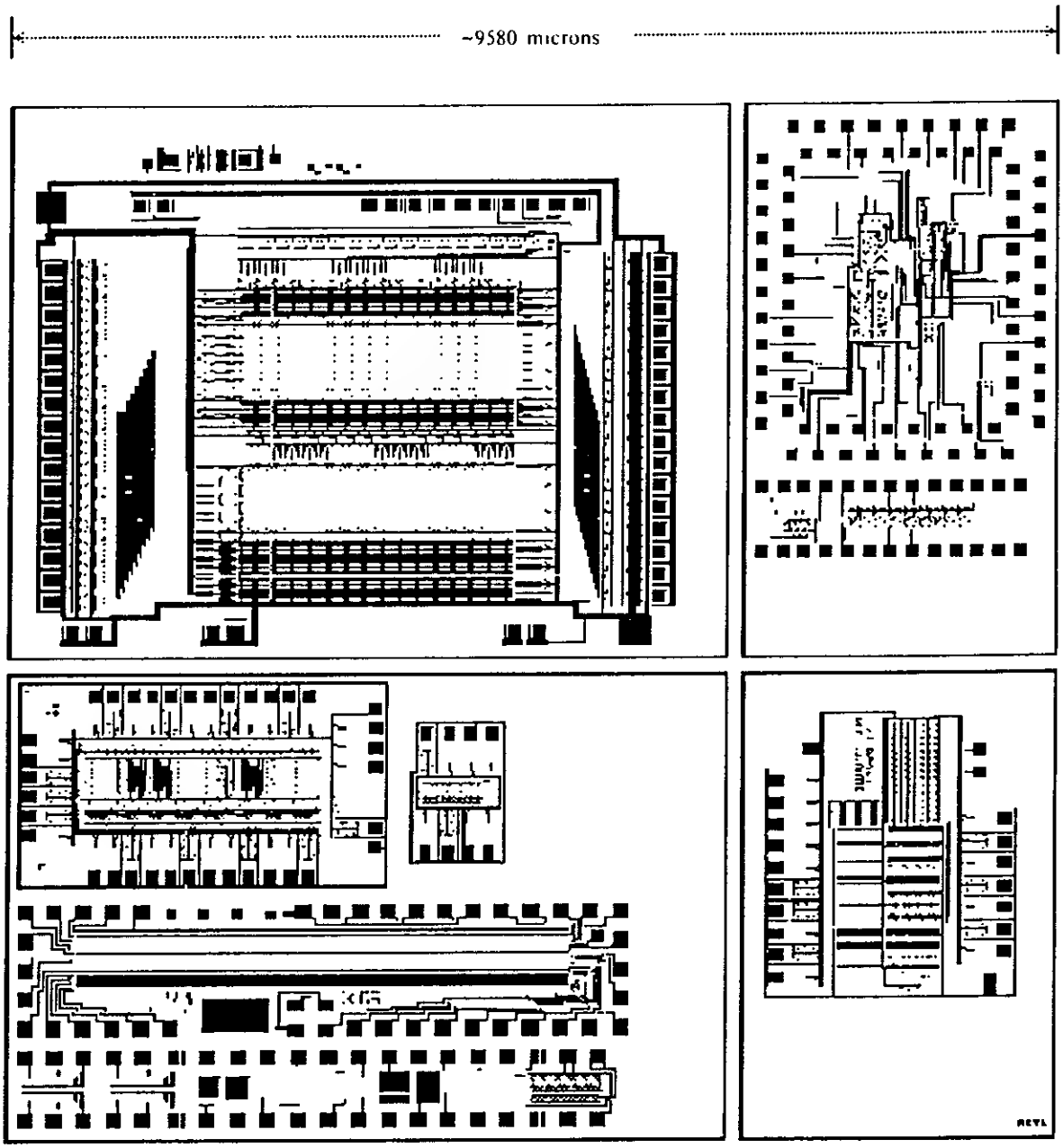


Fig. 21. Collaborative Xerox PARC/Caltech Multi-Project Chip

[Software Blowback from the PG File of the Metal Layer]

different chips, each of which may contain more than one project. When masks are made, each reticle is photorepeated at intervals in x,y corresponding to its outer dimensions minus some scribe line overlap. In the example in figure 21, the x,y stepping distances were both ~9700 microns. Fabricated wafers are scribed and diced on all scribe lines, including the interior ones, to yield chips of typical sizes. One of the projects, on the lower left chip in figure 21, is an experimental charge coupled device array [R. Davies]. The CCDs rode along on this chip set to obtain working masks for use in a completely different process technology (triple poly) from the standard nMOS the other projects used.

Figure 22 provides a higher magnification PG file software blowback of the region near the center of the left scribe line of the chip set. Alignment marks and line width testers (C/D's) were placed in this region, as noted in the figure. Software blowbacks of individual mask levels, more closely resembling the reticles and masks than would a composite design checkplot of all levels, are useful in conveying such location information to the mask and fab houses. Parity marks were not needed on the reticles for this project chip set. Fiducial marks were placed on the reticles by the mask house. Since the software converting the design files to PG files had just been constructed prior to organizing this chip set, reticle blowbacks were requested before proceeding further with maskmaking to verify that everything through pattern generation had worked correctly.

Some other practical details: Participants in the chip set shared some of the commonly used layout items normally required in any project. Examples were input contact pads with attached "lightning arrestor" circuits to protect the input MOSFET gates, and output drivers snaked around and attached to output pads. Even at current device sizes, pads occupy a large fraction of the chip area for large collections of projects, and participants tend to make the pads as small as their bonding skill allows. A square pad ~75μm on a side is a rather small bonding target, and 125μm on a side is easier for the novice to hit. Perhaps ~100μm square pads separated by ~75μm is a good compromise, and these should be at least 25μm from any other metal lines to avoid shorting the lines when bonding.

The scribe lines on this chip set were laid out as 140μm wide cuts down to 160μm wide paths on the diffusion level, to provide lanes free of oxide for scribing or sawing. Metal paths 30μm wide were then laid out straddling the boundaries of these scribe lines, to provide electrical contact from the substrate to the metal during the etching of the metal layer. Since all the projects on this chip set were prototype designs, and were not intended to be placed in extended use, the chips were not overglassed. Eliminating the overglassing

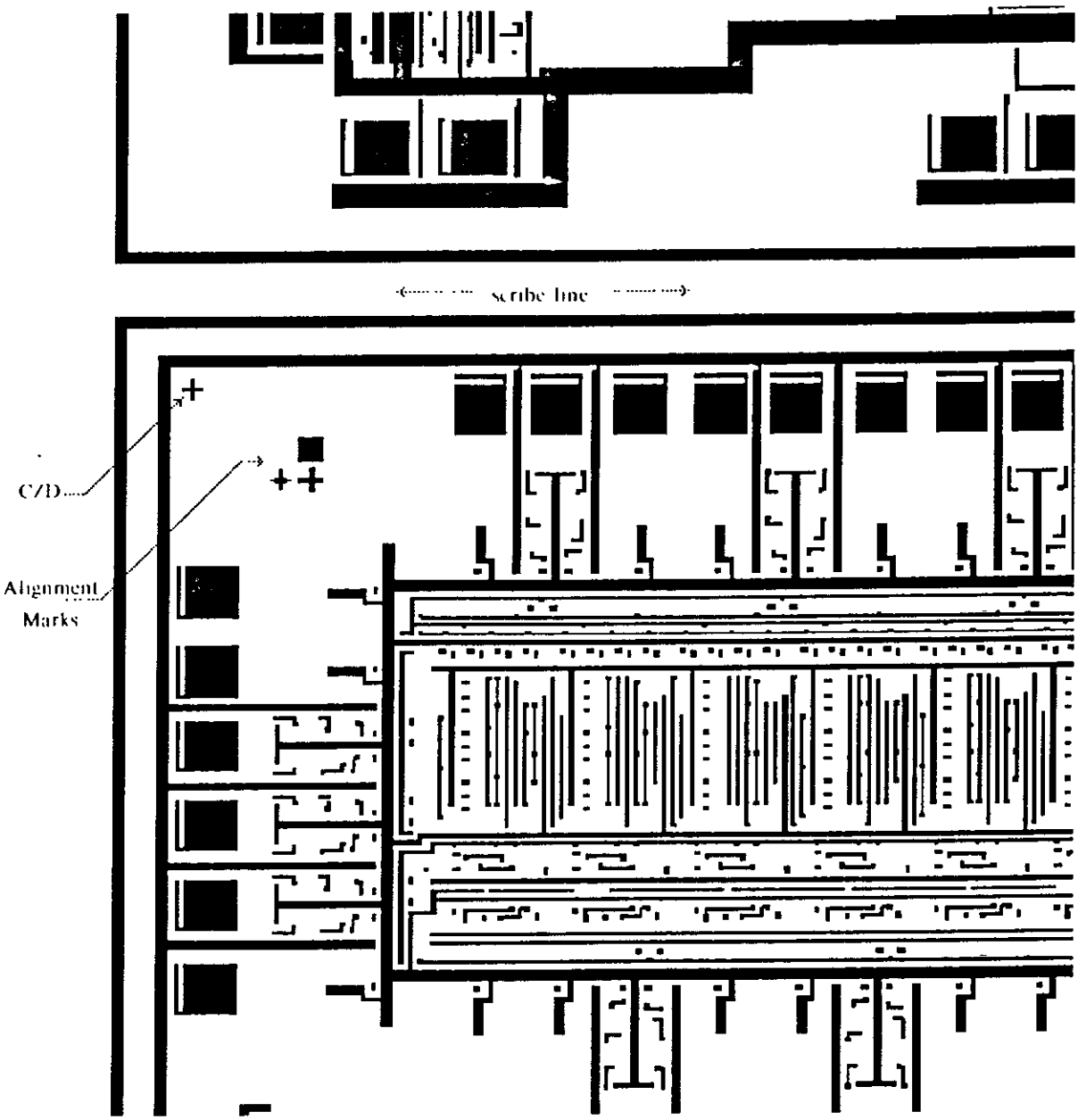


Fig. 22. Close-Up View of Figure 21 Region Containing Mask & Fab Information

(xcloseup press)

meant that a mask level for defining cuts through overglassing over the contact pads and scribe lines was not needed, reducing maskmaking costs. On the other hand, the chip set included a mask level to pattern the thin gate oxide, to provide buried contacts between diffusion and poly that do not require metal coverage as does the butting contact. Such buried contacts enable more compact layouts, but are subject to a rather complex set of design rules, require an extra mask level, and sometimes reduce yield and reliability.

Deleting the overglassing process step also made it possible to electrically probe interior points on the chips during testing, probing small metal test pads included in the layouts. Such pads must be placed with care, however, because they hang relatively large capacitances onto circuitry and slow it down. Note that test pad probing requires special jigs and a stereo microscope, and that it is only possible to directly probe the metal layer. Testing uncovered chips may also require reduced light levels. The operation of dynamic circuits, i.e. those which use a pass transistor input into a gate having no other electrical connection, can be severely affected by light. Light induces leakage currents in the n-p junction between source and drain regions and the substrate. At room temperature, charge stored on dynamic nodes can be retained for many milliseconds in the absence of light. However, in normal room light the retention time is reduced to tens of microseconds. Thus care should be taken to avoid high light levels when long clocking periods are used. Dynamic memory chips are packaged in opaque black packages because of this effect.

A software blowback of the metal mask PG file of another project set, organized at Caltech, is shown in figure 23. The total area of this multi-project chip set is ~ 1 cm^2. It is subdivided into four major sections: The lower right quadrant contains the OM2 Data Engine described in Chapter 5, layed out using $\lambda = 2.5\mu m$. The upper right quadrant contains a 16 by 16 bit multiplier with on-board accumulator [by Rod Masumoto, Caltech], also using $\lambda = 2.5\mu m$. The lower left quadrant contains a subsystem, laid out using $\lambda = 2.9\mu m$, which converts output from one port of a computer memory into the red, green, and blue analog signals for driving a color TV monitor. The upper left quadrant contains 28 projects, mostly from students in an LSI Systems course at Caltech. Other small projects are located along the left edge of the multiplier, and in the unused area within the TV subsystem project. The source material for this project chip set was generated on three different computer systems, in two different languages. Check plotting and viewing were done on three other systems. In addition to the Caltech projects, this chip set contains projects from Carnegie-Mellon University, Washington University (St. Louis), University of California, Irvine, and the Jet Propulsion Laboratory. Approximately 500,000 pattern generator

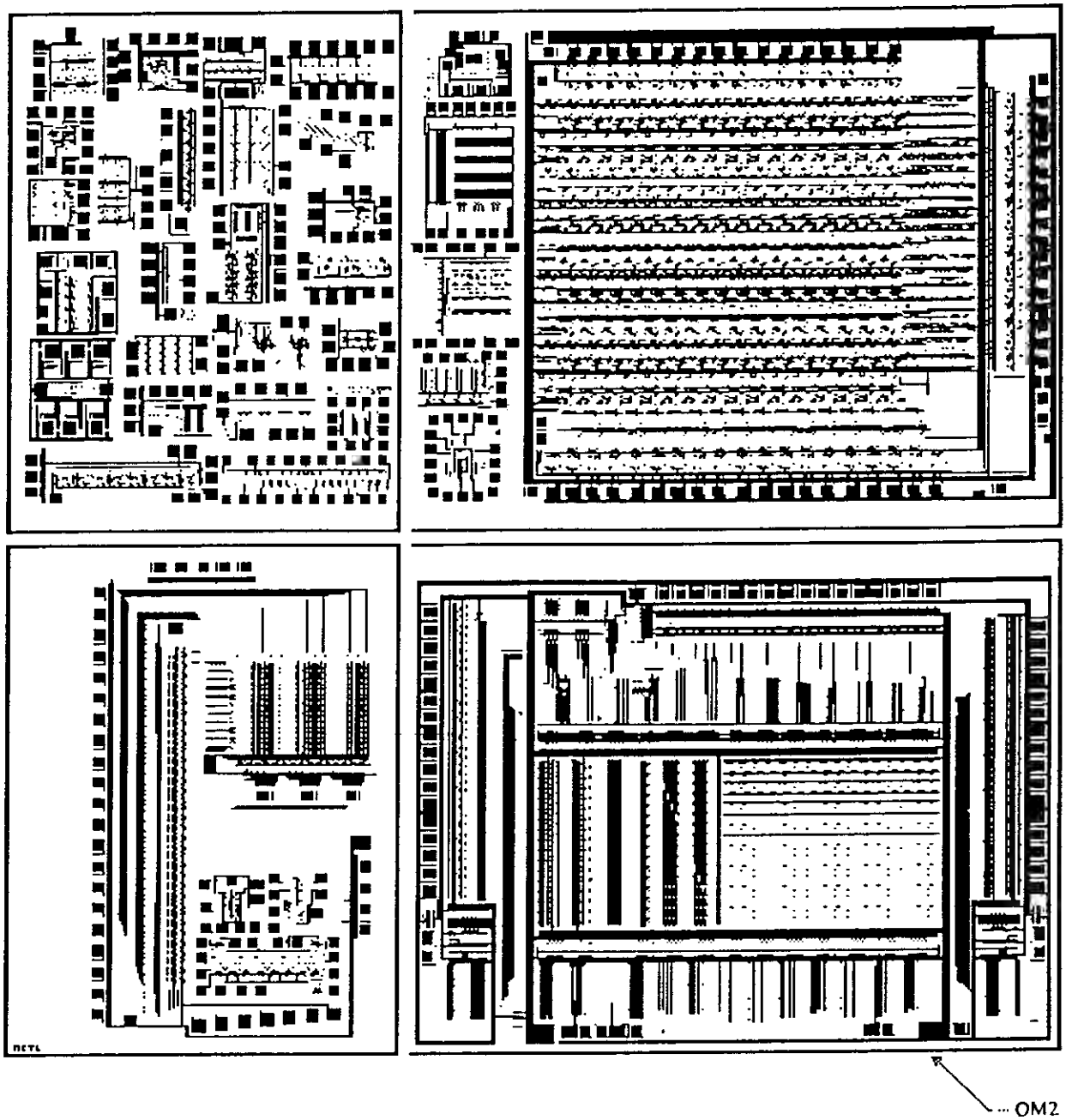


Fig. 23. Multi-Project Chip Set Organized at Caltech

rectangles were required to pattern the reticles for the five mask levels used in this project set. Conversion from intermediate form to PG files required ~10 CPU hours on the Caltech DECsystem 20.

The masks for the multi-project chip sets shown in figures 21 and 23 were produced by Silicon Valley mask houses from PG tapes, accompanied by PG file software blowbacks showing the locations of auxiliary layout items to be used during implementation, and by spec sheets containing a list of mask and fab specifications and parameters. These spec sheets contain two types of information:

(i) that which the mask house will need for reading the PG tape, generating the reticles, and stepping the master masks. This includes whether dimensions are in Metric or English units, whether fiducials and parity marks have been laid out or are to be placed by the mask house, desired reticle magnification (usually 10X, sometimes 5X), the x,y step and repeat distances, the type and magnification of reticle blowbacks desired, and whether maskmaking beyond reticle generation is to be contingent upon blowback inspection. This information is independent of the chosen fab line.

(ii) that which is specific to the fab line, or lines, on which the wafers will be fabricated. Examples here are the number, size, and type of working plates desired, and the photographic polarity of the working plates, i.e. whether they are a positive or negative image of the PG pattern. The polarity of the working plates depends on the process step and on whether positive or negative resist is used. In addition, it is customary to specify how much, if any, the lines in the image will be expanded or contracted to compensate for growth or shrinkage of regions due to the process. This so-called "pulling" of line widths in maskmaking may begin as far back as at pattern generation. Thus, while the patterning and fabrication processes are design and layout independent, they are usually coupled, and masks made for a run on one fab line are not necessarily useable elsewhere.

Maskmaking and patterning technology will remain in a state of transition for years to come. The present shift is from contact printing with working plates to projection alignment using original master masks. These two alternatives are illustrated in figure 24. From the system designer's point of view, at the interface to the mask and fab firms, they present no essential differences, requiring perhaps slightly different specs, and yielding different intermediate artifacts. In the next section we discuss the future evolution of these technologies, presenting several implementation schemes likely to become commonplace over

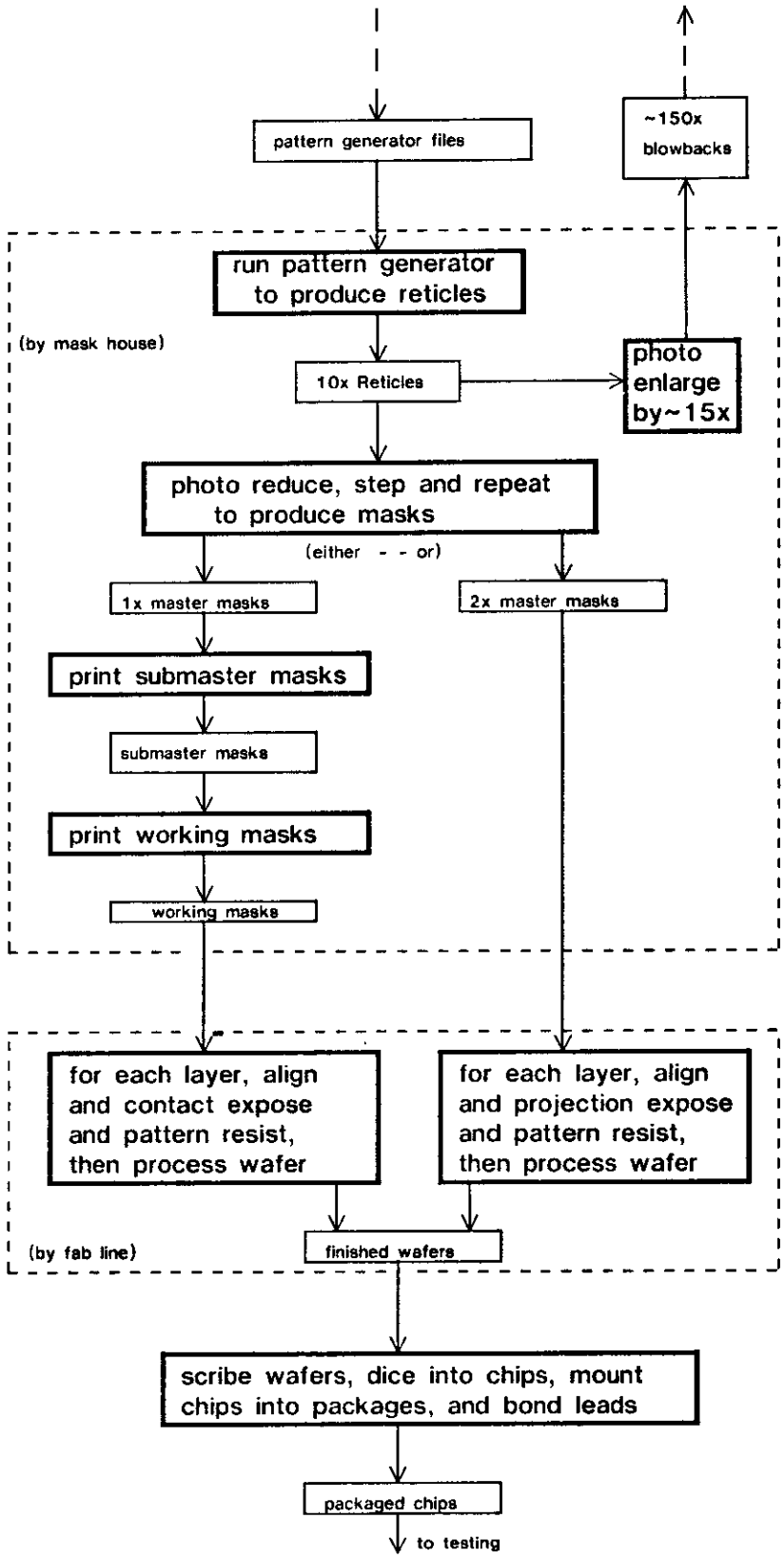


Fig. 24. Present Maskmaking and Fabrication: Two Alternatives

(prespf sil)

the next decade. These schemes will enable fabrication of systems much denser and faster than present ones. However, the basic concepts of the design methodology will still apply. Remembering our film processing analogy, we will have "finer grain" and "faster" film available as time passes. However, the basic art of photography remains.

Patterning and Fabrication in the Future

As λ is scaled down toward its minimum value, ultimately limited by the physics of semiconductors to about 0.1μm, it will become feasible to implement single chip, maximum density VLSI systems of enormous functional power. Patterning and fabrication at such small values of λ requires that certain fundamental problems be overcome[4]. In this section we will discuss alternative solutions to two of the major problems: At values of λ of ~2 μm, a problem of *runout* is encountered, causing successive patterning steps to misalign over large regions of the wafers. This problem is solved by using less than full wafer exposure. At values of λ under 0.5 μm, the wave length of light used in photolithography is too long to allow sufficient patterning resolution. This problem is solved by using non-optical lithography, exposing the resist with electron beams or x-rays.

Historically, silicon wafers have been patterned using full wafer exposure, i.e. using masks which covered the entire surface of the wafer. The pattern for one layer of one chip is stepped and repeated during the fabrication of the mask itself, so that the mask contains the patterns for a large array of chips. During the fabrication of each successive layer on the wafer, that layer's mask is aligned at two points with the pattern already on the wafer, and the entire wafer then exposed through the mask. In the future, as feature sizes are scaled down, full wafer exposure will not likely be possible for reasons developed in this section.

The earliest integrated circuits, circa 1960, were fabricated using wafers of 2.5 cm diameter, and typical chips were 1 to 2 mm, with a minimum feature size of ~25μm. In 1978, production wafers are 7.5 to 10 cm, typical commercially manufactured LSI chips are 5 mm, and minimum feature size is ~5μ. The concurrent development of ever finer features sizes and larger wafer sizes has placed an increasingly severe strain on the process of full wafer exposure. The reasons lie in the physics of wafer distortion.

When a wafer is heated to a high temperature, it expands by an amount determined by the

thermal coefficient of expansion of silicon. A bare wafer will contract exactly the same amount upon cooling, and will therefore remain exactly the same size. Suppose, however, that a layer of SiO_2 is grown on the wafer when it is at the high temperature. The thermal coefficient of expansion of SiO_2 is approximately 1/10 that of silicon. As the wafer is cooled, the silicon will shrink at a rate much greater than that of the SiO_2. Normally the resulting wafer will not be flat, but convex on the SiO_2 side. If the wafer is cooled slowly enough, it is possible to "relieve" the stress induced by the difference in thermal contraction. Wafers in which such stress relief has been achieved are nearly flat but are, of necessity, a different size than they were originally[7a,b].

It might seem that subsequent masks could be scaled to just match the wafer distortion introduced up to the appropriate point in the process. Unfortunately no such correction can be introduced without a knowledge of the pattern of SiO_2 on the wafer. During cooling, dislocations are induced in the underlying silicon crystal at the edges of openings in the oxide pattern. Hence, the magnitude and direction of wafer distortion is dependent in complex ways upon the thickness and distribution of SiO_2 on the surface and upon the details of the thermal cycle. While it is in principle possible to compute a geometric correction for each pattern to be produced, it is clearly not possible to apply one correction for all possible patterns. Misalignment between subsequent layers due to distortion of this type is often referred to as *runout*. Runout due to wafer distortion is today the largest single contributor to misalignment between masking steps. Attempts to use finer feature sizes, which require more precise alignment, on larger wafer sizes, which induce larger distortions, seem doomed to failure unless full wafer exposure is abandoned.

Two attractive alternatives to full wafer exposure are now being explored: (i) electron beam exposure, and (ii) exposure using step and repeat of the chip pattern directly on the wafer.

A scanning electron beam system can be used to expose resist material, and is also capable of sensing a previous pattern on the surface of a wafer. The beam can initially scan an area covering the alignment marks of a particular chip. Information gained from this sensing operation can be used to compute the local distortion, and the chip can be exposed in nearly perfect alignment using these computed values. The process can be repeated for each chip on the wafer, until all have been exposed.

This technique has several virtues. No masks are required. A digital description of the chip can be exposed directly onto a silicon wafer. A different chip can be placed at each chip

location, and this opens up the possibility of greatly extending the multi-project chip concept. However there are also limitations. Data is transferred serially. Even at the highest data rates which can be conveniently generated, a long time is required to expose each chip. More fundamentally, the physics of electron beam interactions places severe restrictions on the minimum practical feature size attainable. When a beam of electrons enters a resist-coated wafer, scattering occurs both in the resist and in the wafer. This backscattering contributes a partial exposure at points up to a few microns away from the original point of beam impingement, and has a number of implications:

(i) The exposure, or spatial distribution of energy dissipation, varies with depth in the resist. Thus resist cross section is not readily controllable.

(ii) Exposure at any particular point depends on all patterns exposed within a few microns. This is known as the "cooperative exposure" or "proximity" effect and necessitates pattern-dependent exposure corrections[8].

(iii) Exposure latitude becomes narrower as the spatial period of a pattern is reduced. This is illustrated in figure 25, which shows the rise in background level exposure as a function of lateral distance for four different spatial periods: (a) 2μm, (b) 1μm, (c) 0.5μm, (d) 0.3μm. The beam diameter is 250 angstrom units, the energy 10keV, the resist thickness 0.4μm. The consequences of this background rise are particularly troublesome for high-speed, low-contrast resists. Experimental results show somewhat greater line broadening than predicted by the model[9].

For the above reasons, the writing time and the difficulty of exposing desired geometries increase rapidly for linewidths below about 0.5 micron[9].

An immediate prospect for achieving feature sizes of 1-2μm with large wafers is offered by stepping the chip pattern directly on the wafer rather than on a mask. This technique avoids the serial nature of the electron beam writing by exposing an entire chip at once. Using good optical systems it has been possible for many years to produce patterns with feature sizes in the range 1 to 2μm. Recent progress in the design of optical projection systems may even make 1/2 to 3/4 micron line width patterns over several millimeter diameter areas practical[10]. Techniques are known for using light to achieve alignments to a small fraction of a wavelength. Recently, an interferometric optical alignment technique has demonstrated an alignment precision of 0.02 micron and should be capable of a

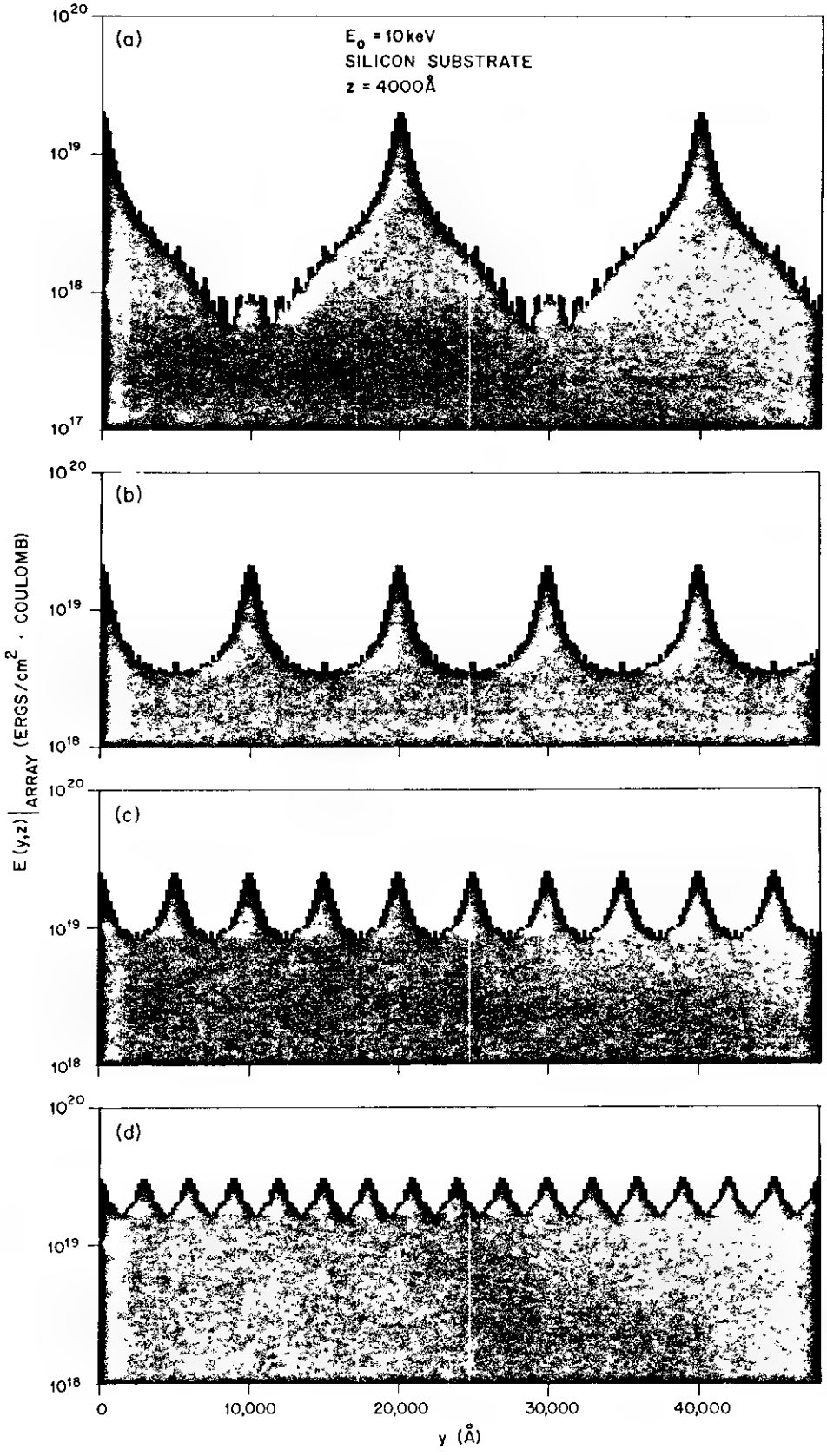


Figure 25. Electron Beam Exposure of Resist on Silicon.
Monte Carlo Calculation of Exposure Level at a Silicon-Resist Interface, as a Function of Lateral Distance, for Four Spatial Periods.
[contributed by H. I. Smith, Lincoln Laboratory, M.I.T.]

reregistration uncertainty less than 0.01 micron[11]. It would seem that devices of ultimately small dimensions (0.25μm) could be fabricated using optical alignment. It must be stressed that a realignment to the underlying pattern *must* be done at each chip location to achieve the real potential of the technique.

The step-and-align technique can be extended to ultimately small dimensions by substituting an x-ray source for the optical one, while retaining the automatic optical alignment system. X-rays require a very thin mask support, e.g. Mylar, upon which a heavy material such as gold or tungsten is used as the opaque pattern. Interactions of x-rays with matter tend to be isolated, local events. No back-scattering of the x-rays occurs, and electrons produced when an x-ray is absorbed are sufficiently low in energy that their range is limited to a small fraction of a micron. For this reason, patterns formed by x-rays in resist materials on silicon wafers are much cleaner and better defined than those attainable by any other known technique (see figure 26). X-rays of very high intensity can be efficiently obtained from the synchrotron radiation of an electron storage ring. The time required for exposing a chip with such a source is no more than that required at present using optical exposures. Both optical and x-ray techniques have the property that the total *exposure* time per wafer can be made independant of how much of the wafer is exposed at a step. Therefore, the only penalty in a step and align process is the time required for mechanical motion and alignment.

It appears that we have in hand all of the techniques for ultra fine line lithography, even on larger silicon wafers. Both electron beam and optical stepping work must, however, focus on local alignment as the crucial step in achieving high density, high performance LSI.

We now describe a production lithography system for ultimately small dimensions. A major component of the system is a 500 to 700 MeV electron storage ring, approximately 5 meters in diameter, shaped in the form of a many sided polygon. The electron beam within this storage ring is deflected at each vertex by a superconducting magnet. This deflection results in a centripetal acceleration of the electrons, and hence in an intense tangential emission of synchrotron radiation. The most important component of such radiation is soft x-rays in the 280 to 1000 eV quantum energy range (wavelengths of 0.004 to 0.001 μm). Such x-rays are ideal for exposing resist materials with line widths in the 0.1μm range[12,13].

One exposure station is fitted to each vertex of the storage ring. Each exposure station has an automatic optical alignment system for individual alignment of each chip[11]. Coarse

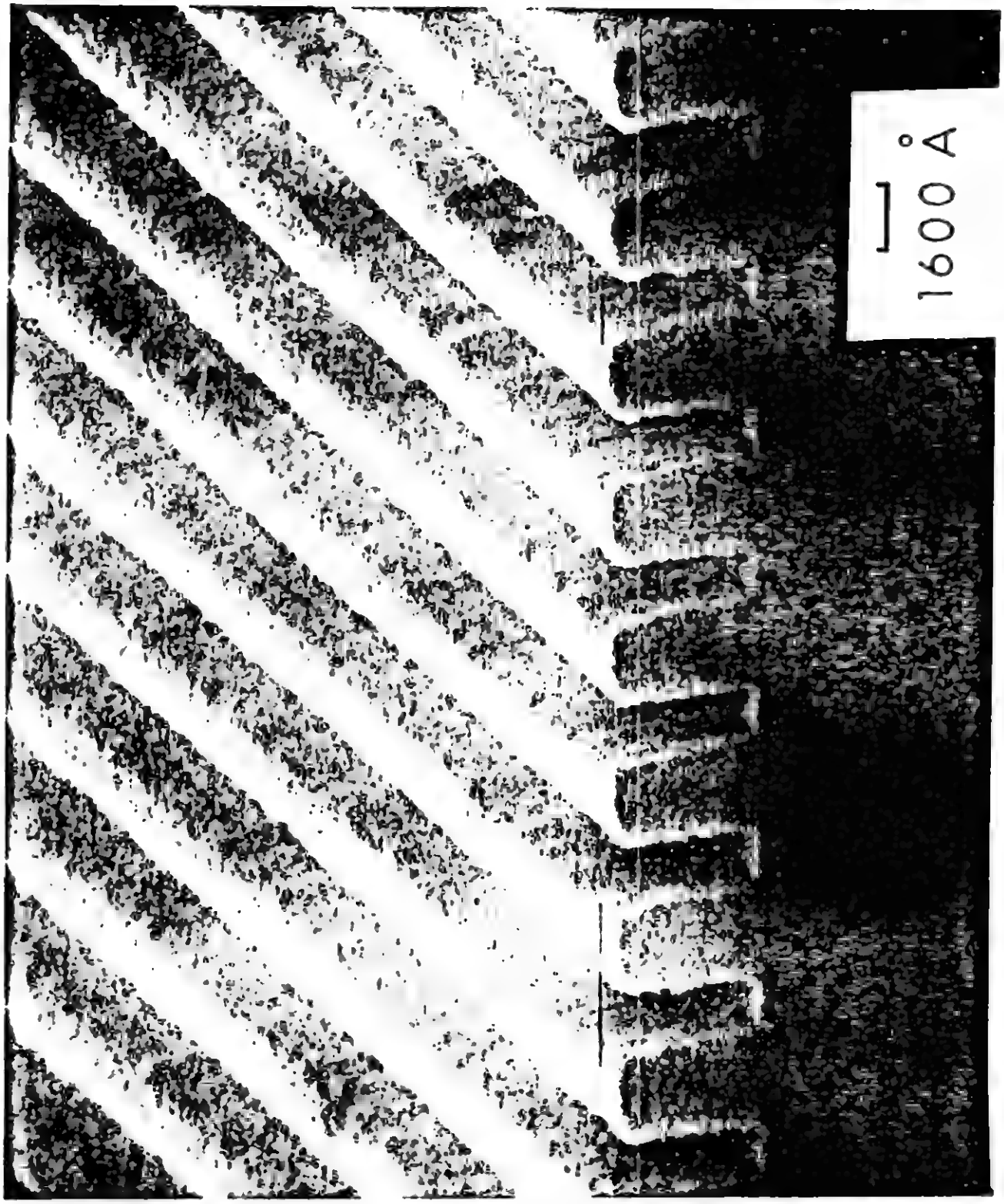


Figure 26. Resist on Silicon, Patterned by X-Ray Exposure.
[contributed by H. I. Smith, Lincoln Laboratory, M.I.T.]

alignment is controlled by a laser interferometer and the wafer brought into position by ordinary lead screws moving a conventional stepping stage such as those in current photorepeaters. Auxiliary alignment features are placed on each mask level within each chip. Misalignment of two such patterns on the wafer relative to those in the mask produces Moire patterns which are detected by photosensors and fed to a computer system. Piezoelectric transducers driven by the computer system bring the wafer into final alignment under the mask. Each exposure station in such a system is capable of aligning and exposing one layer of one chip every few seconds. Each chip may contain of the order of 10^7 devices, which is the equivalent of several *wafers* at today's scale.

An overview of the possible routes from design files to finished chips with sub-micron layout geometries is shown in figure 27. In the immediate future, alignments much better than those achievable today will be possible with the optical step and align technique (leftmost path in figure 27). In addition, this scheme eliminates the step and repeat process in mask making, enabling considerably shorter turnaround time. The rightmost path, direct electron beam writing on the wafer, promises the ultimate in short turnaround time. It can be viewed as using the fab area as a computer output device. For high volume manufacturing, at ultimately small dimensions, the center path as described above will most likely become the workhorse of the industry.

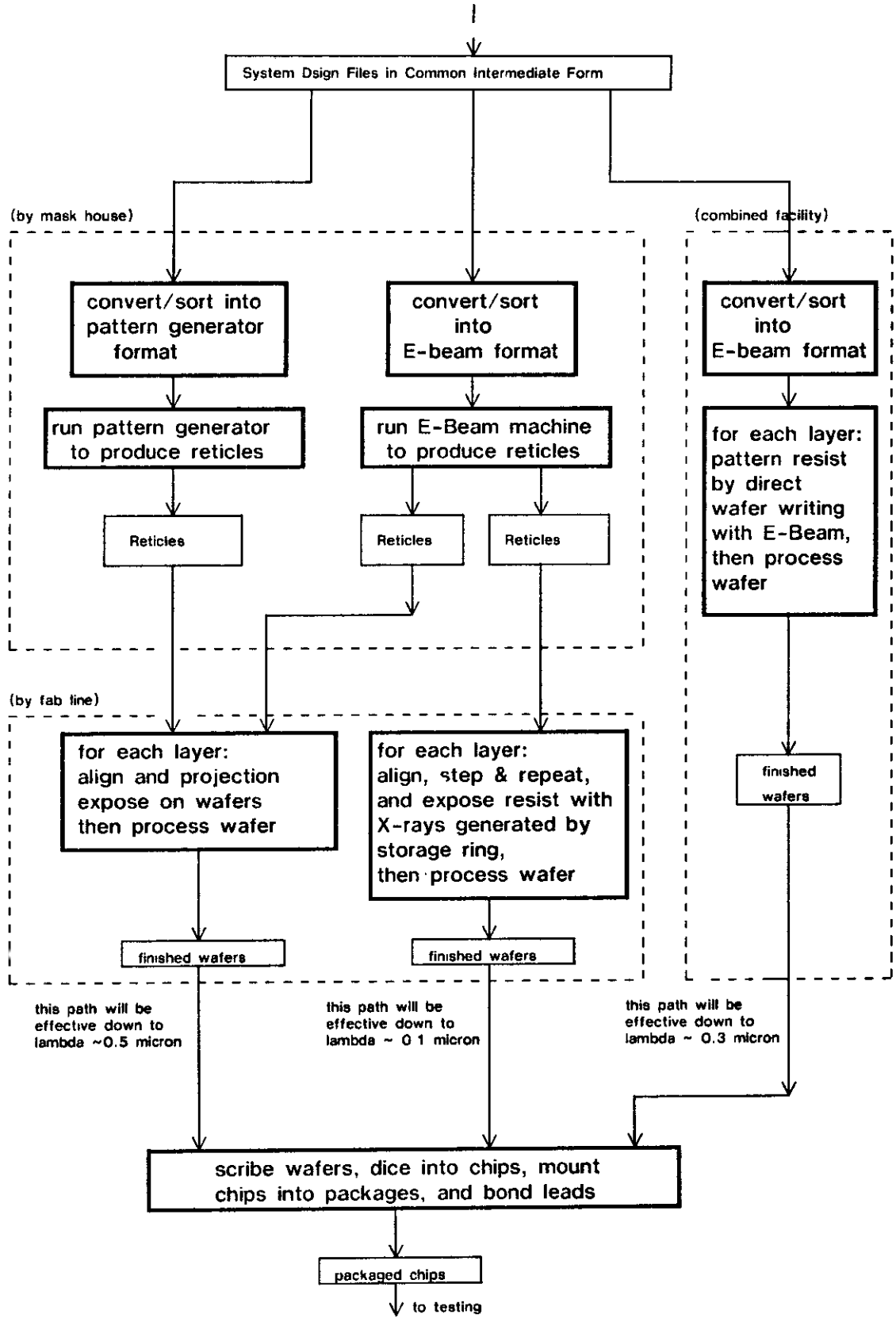


Fig.27 Some Future Patterning and Fabrication Alternatives

(futpf sil)

Fully Integrated, Interactive Design Systems

{ in preparation }

- - - creating a data structure which allows the various levels of interactive processes to operate on the same data base - - - nodes, transistors, cells, and instances - - - operations on the data base - - - interactive logic transfer function tests - - - interactive circuit transfer function tests - - - interactive design rule checking - - - the filing problem - - -

System Simulation, Test Generation, and Testing

{ in preparation }

- - - system-level/register-transfer-level design description and simulation - - - testing the system design - - - practical strategies for structured VLSI system development - - - designing for testability - - - generation of test sequences - - - testing the chips - - -

References

1. D. G. Fairbairn, J. A. Rowson, "ICARUS: An Interactive Integrated Circuit Layout Program", submitted to 15th Design Automation Conf., IEEE, June 1978.

2. J. A. Rowson, "A Data Structure for Interactive Integrated Circuit Design", submitted to 15th Design Automation Conf., IEEE, June 1978.

3. A. C. Kay, "Microelectronics and the Personal Computer", Scientific American, Sept.1977, pp 210-228.

4. I. E. Sutherland, C. A. Mead, T. E. Everhart, "Basic Limitations in Microcircuit Fabrication Technology", ARPA Report R-1956-ARPA, November 1976.

5. C. A. Mead, "Ultra Fine Line Lithography", Display File #1179, Dec. 2, 1977, Department of Computer Science, California Institute of Technology.

6. GCA/D.W.Mann, "3600 Software Manual, Appendix B: 3600 Pattern Generator Mag Tape Formats", GCA Corporation, IC Systems Group, Santa Clara, Ca..

7a. I. A. Blech, E. S. Meieran, "Enhanced X-ray Diffraction from Substrate Crystals Containing Discontinuous Surface Films", J. App. Phys., vol. 38, pp. 2913-2919, June 1967.

7b. E. S. Meieran, I. A. Blech, "High Intensity Transmission X-ray Topography of Homogeneously Bent Crystals", J. App. Phys., vol. 43, pp. 265-269, Feb. 1972.

8. M. Parikh "Self Consistent Proximity Effect Correction Technique for Resist Exposure", to be published, J. Vac. Sci. Tech. Jan./Feb. 1978.

9. R. J. Hawryluk, H.I. Smith, A. Soares and A. M. Hawryluk, "Energy Dissipation in a Thin Polymer Film by Electron Beam Scattering: Experiment", J. Appl. Phys., vol. 46, pp.2528-2537, June 1975.

10. J. S. Wilczynski, "A Step and Repeat Camera for Direct Device Fabrication", Semicon East, Sept. 1977, Boston; M. Huques, M. Babolet "Lenses for microelectronics", and P. Tigreat, "Use in a Photodemagnifier", International Conference on Microlithography, Paris, June 1977.

11a. D. C. Flanders, H. I. Smith, S. Austin, "A New Interferometric Alignment Technique", App. Phys. Lett., vol. 31, pp. 426-428, Oct. 1977.

11b. S. Austin, H. I. Smith, D. C. Flanders, "Alignment of X-Ray Lithography Masks Using a New Interferometric Technique - Experimental Results", to be published, J. Vac. Sci. Tech., Jan./Feb. 1978.

12. B. Fay, et al.,"X-Ray Replication of Masks Using the Synchrotron Radiation Produced by the ACO Storage Ring", App. Phys. Lett, vol. 29, pp. 370-372, Sept. 1976.

13. E. Spiller, et al., "Application of Synchrotron Radiation to X-Ray Lithography", J. App. Phys., vol. 47, pp. 5450-, Dec. 1976.

14. N. Wirth, "What Can We Do about the Unnecessary Diversity of Notations for Syntactic Definitions?", Communications of the ACM, Nov. 1977.

Reading References

R1. J. J. Donovan, "Systems Programming", McGraw-Hill, 1972, an introductory text on this subject, provides practical information on the implementation and use of assemblers, macro-processors, compilers, loaders, operating systems, etc.

R2. W. M. Newman, R. F. Sproull, "Principles of Interactive Graphics", McGraw-Hill, 1973, is the classic text on interactive computer graphics, and is recommended reading for those interested in constructing interactive layout systems and fully integrated, display oriented, design systems.

R3. P. Freeman, "Software Systems Principles", Science Research Associates, Inc., 1975, presents the basics of a broad range of topics important in the building of software systems.

Chapter 5: The Design of a Data Processing Engine

Sections:

The Overall Structure - - - The Arithmetic Logic Unit - - - ALU Registers - - - Buses - - - Barrel Shifter - - - Register Array - - - Communication with the Outside World - - - Machine Operation Encoding - - - Functional Specification of the Machine

Up to this point, we have chosen simple examples to illustrate the fundamental properties of integrated systems, and the type of design methodology which can be used to build hierarchically organized, complex systems. In order to more fully clarify some of these techniques, we will now study the design of a simple data processing structure: the data path from a microprogrammed 16-bit machine, undertaken as a university project in experimental computer architecture. The "Our Machine" (OM) project was started in 1976 by Carver Mead as part of the LSI Systems course at Caltech. Early contributions were made by Mike Tolle [Litton Industries], while attending this course. Other participants were Caltech students Dave Johannsen and Chris Carrol, with much inspiration from Ivan Sutherland. By December 1976, a first design (OM_0) was nearly completed. The participants decided at that time that the design had become "baroque" and ugly, and it was scrapped. A new design (OM1) was completed by March 1977 by Dave Johannsen, Chris Carrol, and Rod Masumoto. Fabricated chips were received in June 1977. It was this chip which appeared in the September 1977 Scientific American article by Sutherland and Mead. The chip was fully functional except for a timing bug in the dynamic register array (which had been designed in departure from the structured design methodology developed in this text). A complete redesign of the chip was undertaken in June 1977, by Dave Johannsen. By September, a complete set of new cells had been constructed, and the design was completed by December. Cells from this chip and its companion, the controller chip described in chapter 6, were used as examples in chapter 3. The redesign included improvements in the encoding of the microcode control word, and rigorously applied the structured design methodology.

The chapter is presented in two separate parts. The first part outlines the architectural requirements for the chip, and illustrates how the design methodology was applied to satisfy them. The second part is a precise functional description of the chip, intended as a user manual for those who microprogram the machine. A more complete discussion of the overall system architecture is given in chapter 6.

The Overall Structure

The basic requirements initially established for the machine were that it be gracefully interconnectable into multiprocessor configurations, that it be microprogrammable, so that OP code sets can be configured to the application at hand, that it be able to do variable field operations for emulation instruction decoding, assembly of bit-maps for graphics, etc., and that its performance be as fast as possible.

In order to satisfy the first requirement, it was decided that the machine would initially have two ports: one to be used for a system interconnection, and the other for local memory, I/O, etc. It was perceived that in many systems much time is lost in assembling two operands for most operations, so it was decided that the machine have two internal buses, and that any registers in the machine be two-port registers. The requirement for gracefully handling variable length words required a shifter at least sixteen bits long, and the last requirement dictated an arithmetic logic unit of considerable flexibility while not sacrificing speed. The strategy was adopted initially that the two buses would run through the actual processing array, from one end of the chip to the other. One port was to be located at the left end of the chip, and the other port at the right end, and the two system buses were to run the full length of the chip between the two ports through the actual register and data processing array.

The three main central, functional blocks in the machine were the register array, the shifter, and the arithmetic logic unit. It was decided to run the control lines vertically in metal, and the buses horizontally in polysilicon, and that power, ground, and timing signals would run parallel to the control signals. At this point, it is already possible to make a rather detailed sketch of the general layout of the chip. This arrangement is shown in Fig. 1. The details of these functional blocks will be described in subsequent sections. Included are descriptions of peripheral circuits needed to interface subsystems with each other and to the outside world.

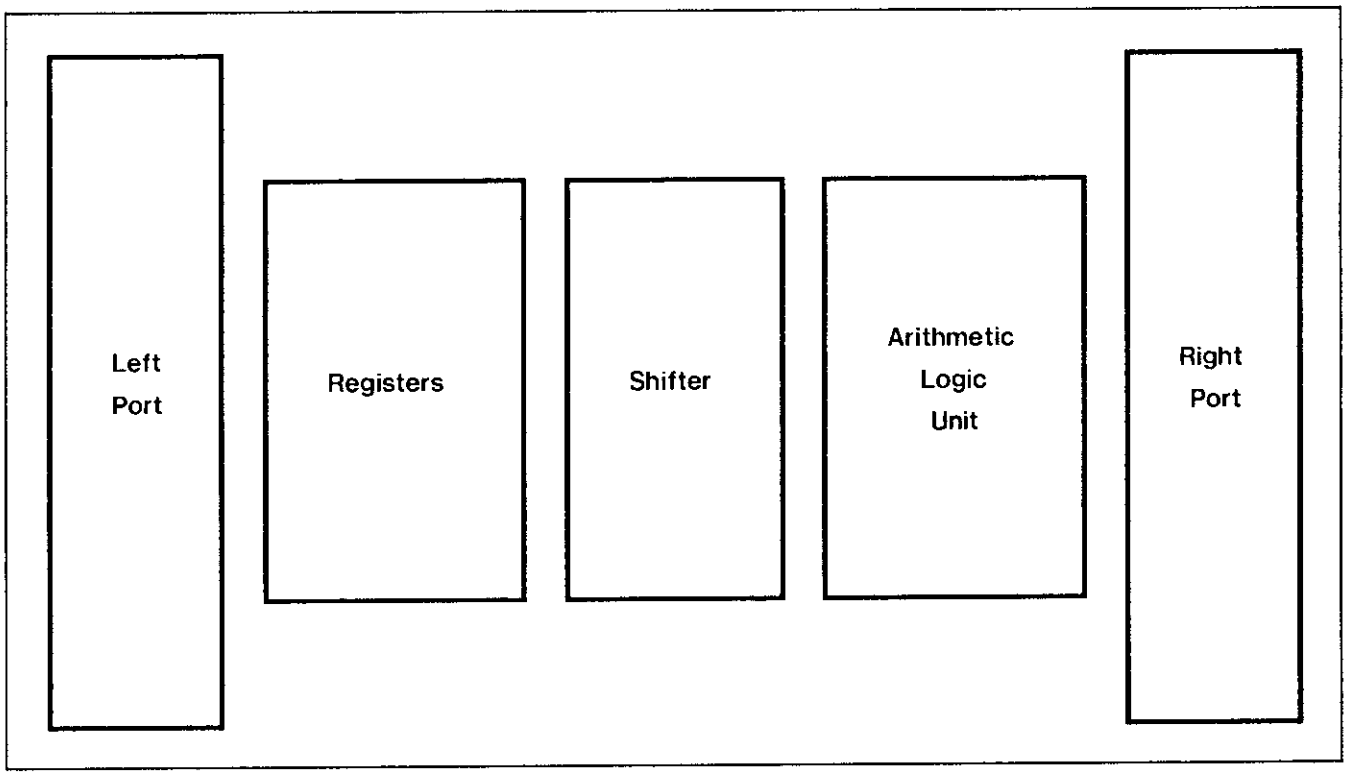


Figure 1. General Layout of a Microprogrammed Data Processing Structure.

The Arithmetic Logic Unit

It was believed that the carry chain would limit the performance of the system and therefore, the carry chain and its associated logic was the first functional block to be designed in detail. Simulations of several look-ahead schemes indicated that they added a great deal of complexity to the system without much gain in performance. For this reason it was decided early in the project to implement the fastest possible Manchester type carry chain (reference 4, chapter 1), similar to that shown in chapter 1, figure 11. The carry chain and its associated logic were allowed to dictate the repeat distance of the cells in the vertical direction. In MOS technology, a Manchester carry chain is particularly limited in its ability to propagate a *high* carry signal. However, it is quite fast in its ability to propagate a *low* carry signal.

In any arithmetic logic unit there will be a null period when the OP code for the next operation is being brought in. Advantage can be taken of this null period to precharge the carry chain and other sections of the machine where timing is particularly crucial. In this way, it is not necessary to propagate high signals through pass transistors where the rise transient would be particularly slow. It was decided to apply this strategy to OM's ALU, and the resulting carry chain is shown in Fig. 2.

The main carry chain runs through the pass transistor from carry-in to carry-out. The carry-in signal is detected by the gate of an inverter which feeds the signal into the subsequent logic of the ALU. Three transistors are used to control the state of the carry-out of each stage. The first one merely precharges the node associated with carry-out during the null period of the ALU. The second is the carry-kill signal which is derived from the inputs to the ALU, and simply grounds the carry-out through a single transistor. The third is a pass transistor which causes carry-out to be equal to carry-in. These last two signals associated with the carry chain in each stage, carry-kill and carry-propagate, are generated by two NOR gates which have kill-bar and propagate-bar as one input and precharge as the second input. Hence, it is assured that the kill signal and propagate signal are disabled during the null period when the precharging takes place.

After some analysis, it was decided that nearly all interesting combinations of carry-in and the input signals could be generated using propagate and carry-in from each stage. Thus the carry-chain itself may be viewed as a logic block with two inputs, carry-kill bar and carry-propagate bar, two outputs, propagate and carry-in, a vertical signal carry-in and carry-out,

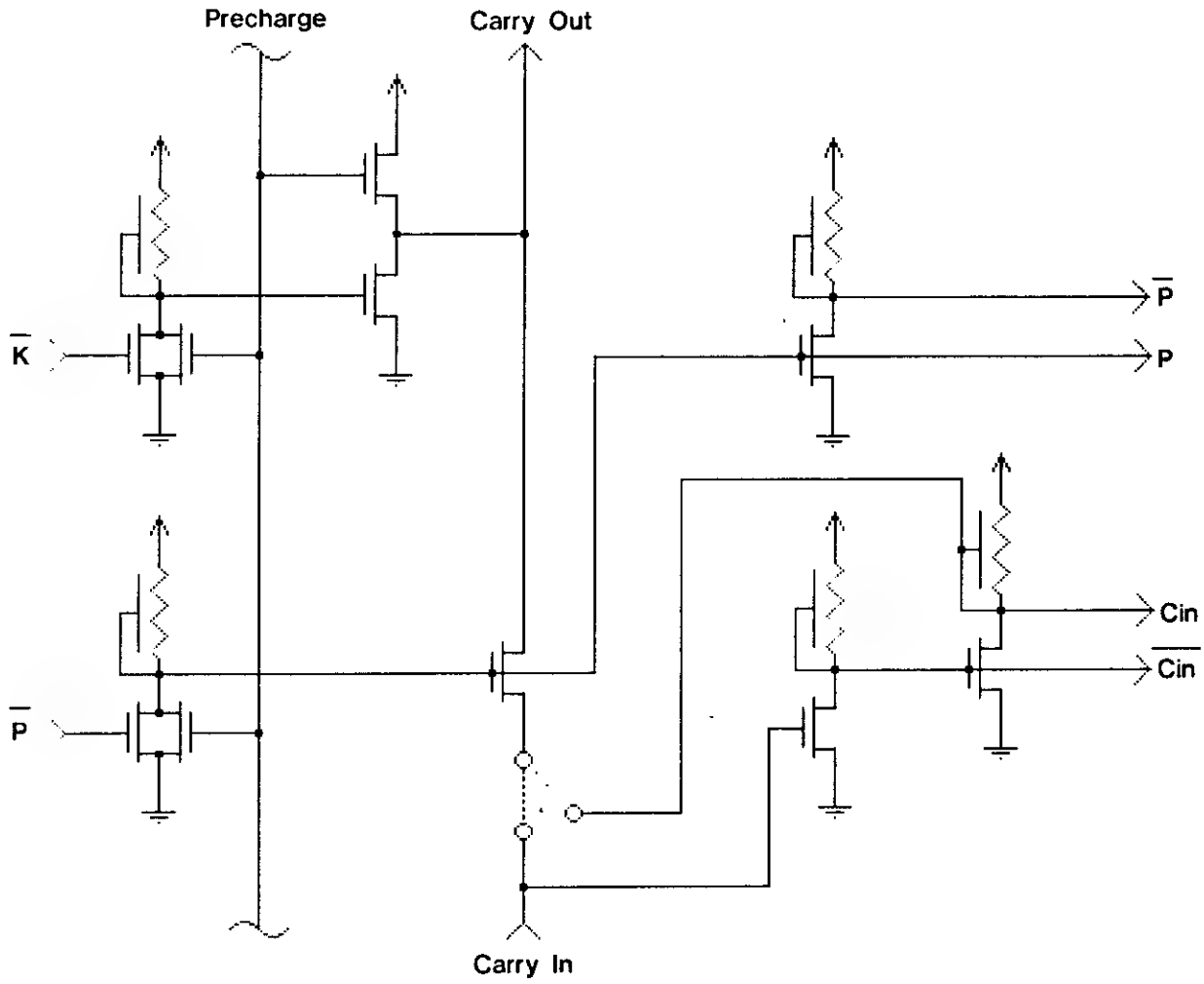


Figure 2. Carry Chain Circuit for the Arithmetic Logic Unit.

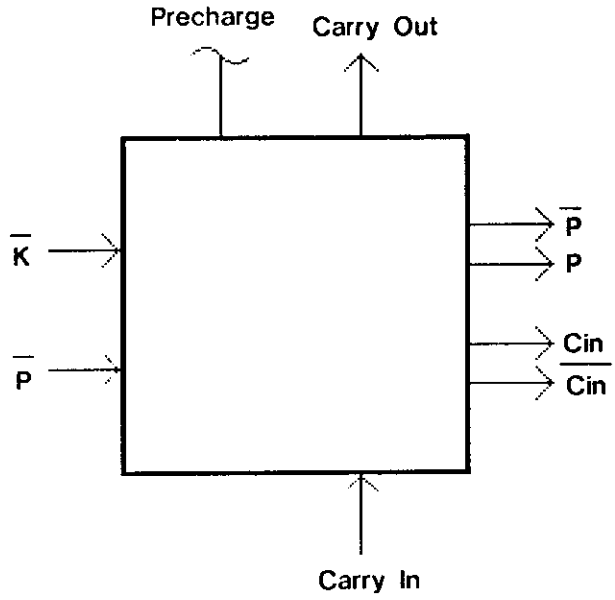


Figure 3. Abstraction of the Carry Chain Circuit.

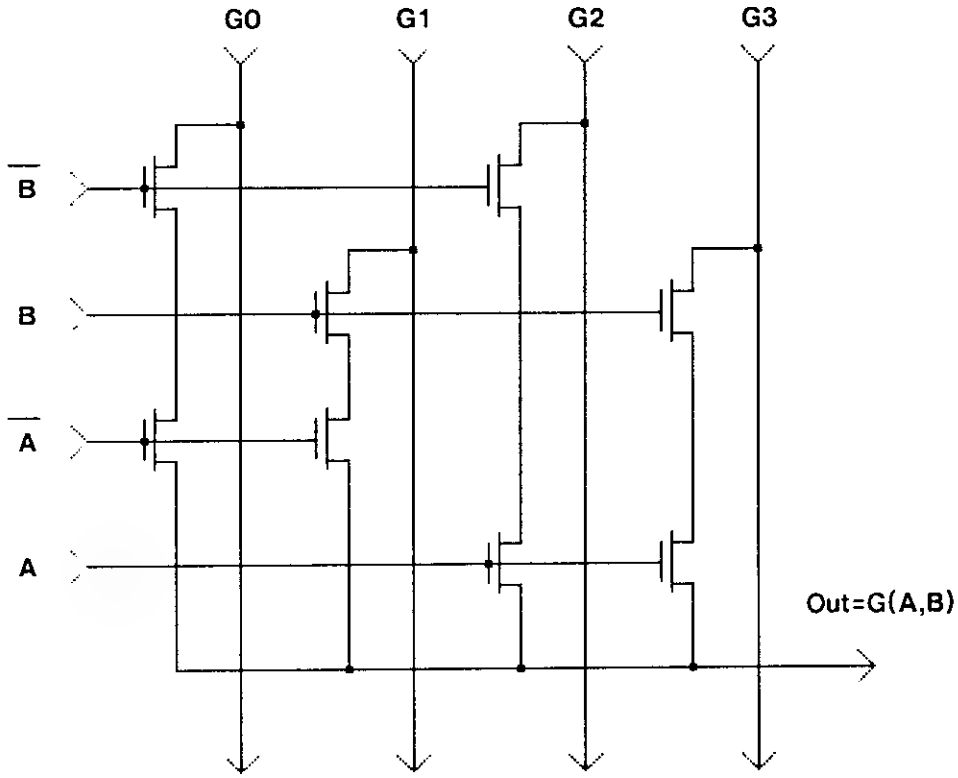


Figure 4. General Logic Function Block Transistor Diagram.

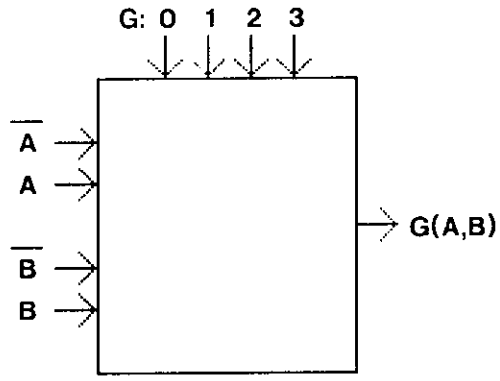


Figure 5. Functional Abstraction of the General Logic Function Block.

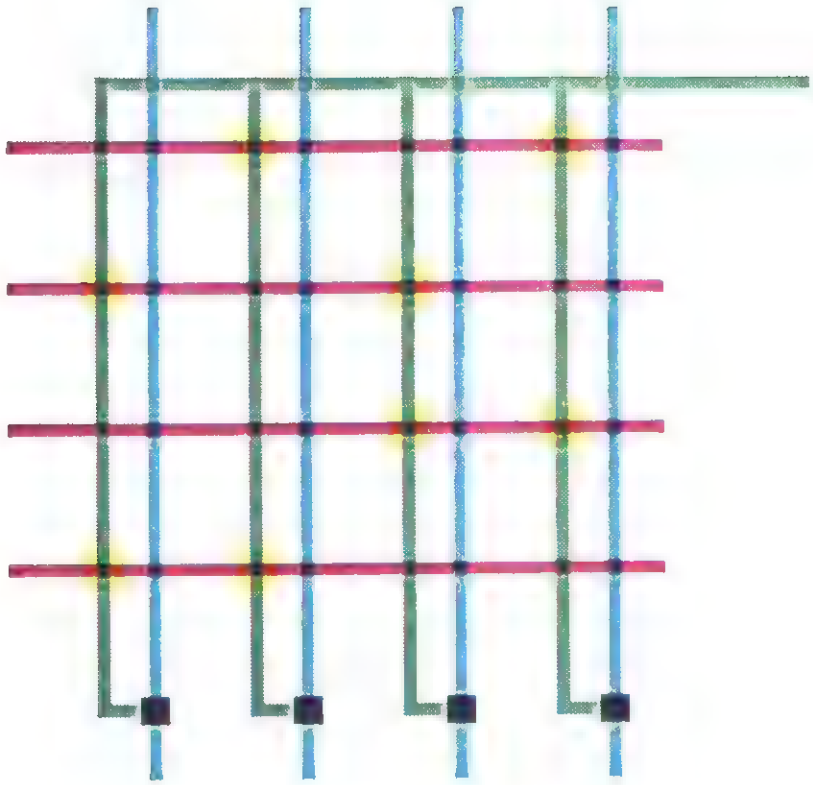


Fig 4a. Stick Diagram of the Function Block

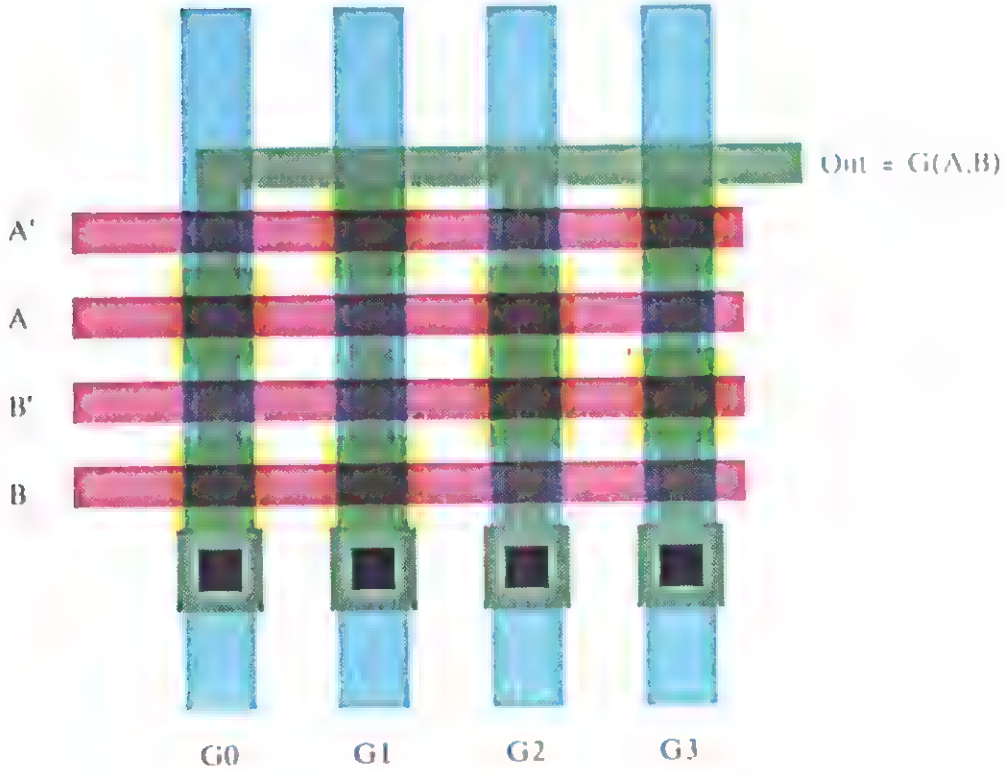


Fig 4b. Actual Layout of the Function Block

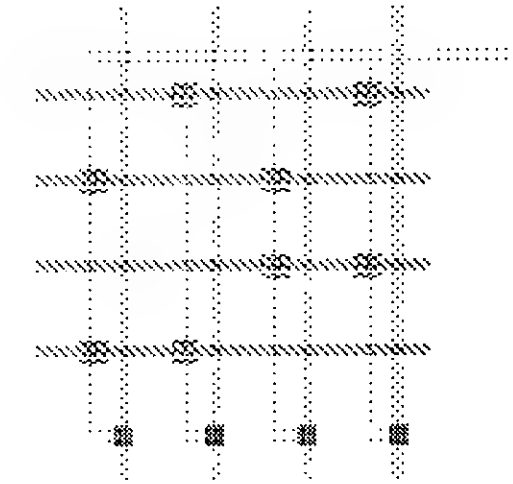


Fig. 4a. Stick Diagram of the Function Block

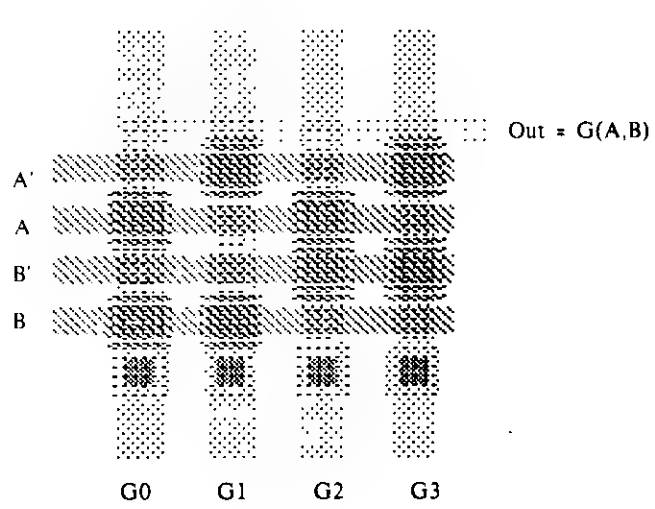


Fig. 4b. Actual Layout of the Function Block

and one control wire, precharge, as shown in Fig. 3.

The task of designing the balance of the ALU is now reduced to that of designing functional blocks to; a) combine the two input variables to form a propagate bar and kill bar, b) to combine carry bar and propagate to form the output signal, and c) drivers for controlling the logical function blocks and deriving a timing for precharge.

A number of random logic implementations of function blocks for deriving kill, propagate, and the output were attempted. All seemed to be at variance with the horizontally microprogrammed architecture of the machine, and required a large amount of area and power. For this reason it was decided to use the general logical function block illustrated in chapter 3, figure 12a. Such circuits are used to generate carry-bar, propagate-bar, and for combining carry-bar in and propagate to form the output. The circuit implements sixteen logic functions of two input variables, and is shown in Fig. 4. It consists of a set of transistors which fully decode the input combination of A and B, and connect one and only one of the vertical control lines to the output, depending on this input combination. Thus, for example, when A and B inputs are both low, the vertical control wire labelled G_0 is connected to the output. The truth table entries for the desired logic function are placed on the G vertical control wires, and the output will then be the desired logic function of the two input variables. For example, if the Exclusive-OR of A and B is desired, a logic-0 will be applied to the control wires 0 and 3, and logic-1 will be applied to control wires 1 and 2. Since it is desired to implement the same logic function on all bits of the word, the control variables G_0 through G_3 need not be generated in every bit slice, but may be generated once at either the top or bottom of the array. The functional abstraction of the circuit of Fig. 4 is shown in Fig. 5.

We are now in a position to form the block diagram for our complete arithmetic logic unit, as shown in Fig. 6. The functional dependence of the output on the two inputs and the state of the carry is determined by a 12-bit number: P_0 through P_3, K_0 through K_3, and R_0 through R_3, together with the carry-in to the least significant bit of the ALU. The ALU is quite general, and its detailed operation set may be left unbound until the control structure of the machine is designed at a later time.

There are two general principles illustrated by this design. First, it is often less expensive in area, time, and power to implement a general function than to implement a specific one. Secondly, if a general function can be implemented, the details of its operation can be left

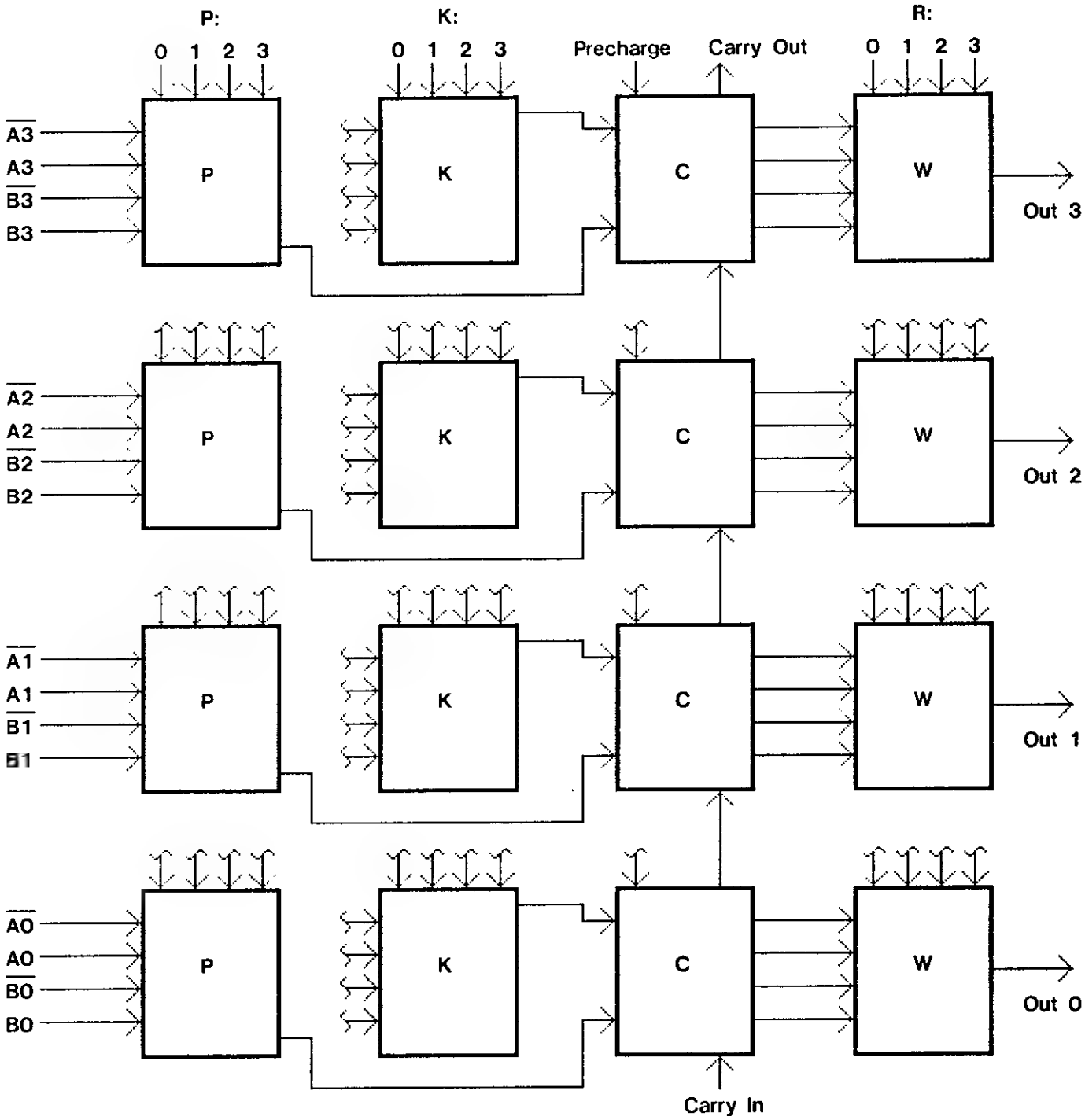


Figure 6. Block Diagram of a 4-Bit ALU.

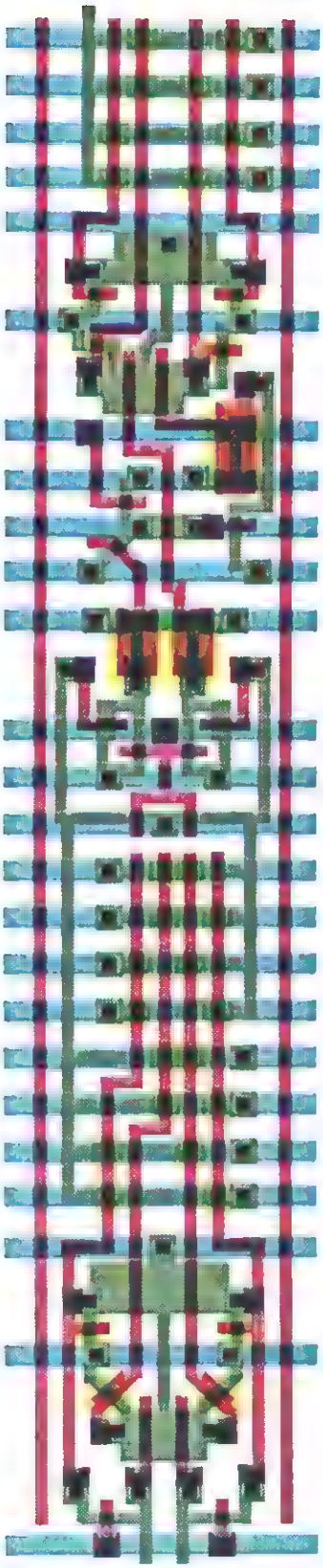


Figure 6a. Layout of ALU and Input Registers

unbound until later, and hence, provide a much cleaner interface to the next level of design. The detailed choices of which functional entities to leave unbound and which to bind early requires a considerable amount of judgment, and is where much of the skill in integrated system design lies.

Two details need to be dealt with before the arithmetic logic unit function block is complete. Drivers are needed for the P, K, and R terms which will generate signals with the appropriate timing. In addition, inverters must be interposed in the carry chain occasionally to minimize the propagation delay through the entire carry chain. The way we have chosen to implement the interposition of inverters is to recognize that each carry chain function block contains two inverters which present at the output carry-in, having been twice inverted from the actual carry-in signal. If we merely substitute this signal for the carry-out signal from the pass transistor, we have doubly inverted in buffered our carry-in and buffered it to minimize the propagation delay. This approach avoids putting spaces between the carry function blocks for inverters. It is illustrated by the dotted connection lines in Fig. 2. In the actual implementation, the connection through the inverters was made in every fourth stage.

Drivers for the P, K, and R terms have the following function: At some time during the null period of the ALU (which we shall call φ_1), an OP code specifying each of the terms arrives at the input to the driver. It must be latched while the ALU itself is being precharged, and then it must be applied to the P, K, and R terms as soon as the ALU is activated. The P, K, and R function blocks are themselves composed of pass transistors, and their outputs are more effectively driven low than high. For this reason, we will precharge the outputs of the P, K, and R function blocks as well as the carry chain itself. This is most conveniently done by requiring that all of the P, K, and R control signals be high during the null period of the ALU. Then, independent of the states of A and B inputs, the outputs will be charged high by the time ALU active period commences. The control buffer which implements this function is shown in Fig. 7.

The OP code is latched through a pass transistor whose gate is connected to φ_1, and the OP code runs into a NOR gate, the other input of which is also φ_1. Thus, the output of the NOR gate is guaranteed to be low during the φ_1 period. The NOR gate output is then run through an inverting super-buffer, so that during φ_1 the output is guaranteed to be high. At the end of φ_1, whatever OP code is present at the input of the NOR gate is transferred

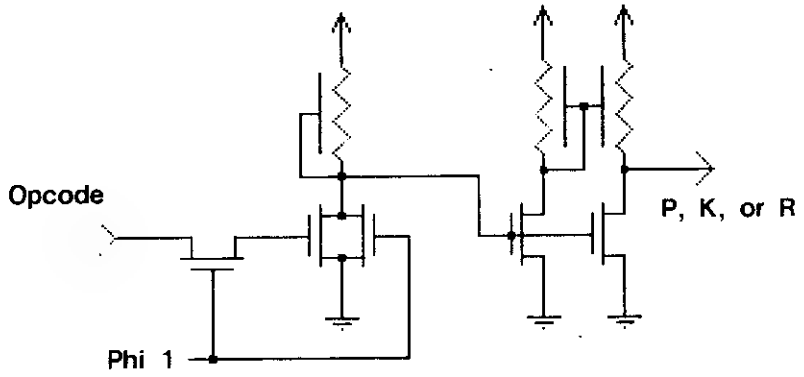
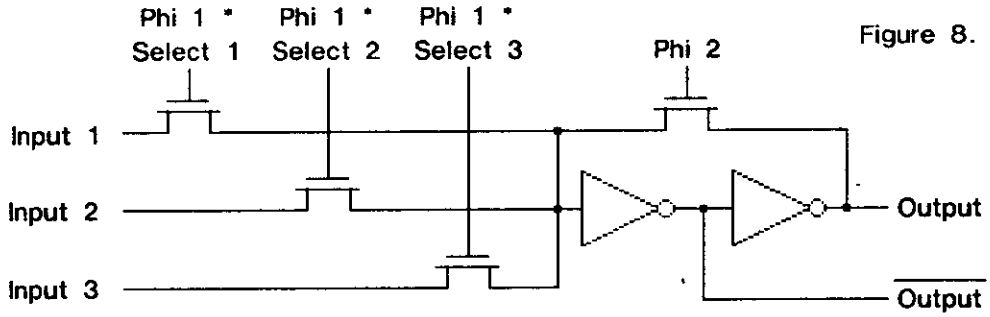


Figure 7. ALU Control Driver

All outputs high during Phi 1
(Precharge)
Selected terms low during Phi 2
Opcode valid during Phi 1



ALU
Figure 8. Input Register.

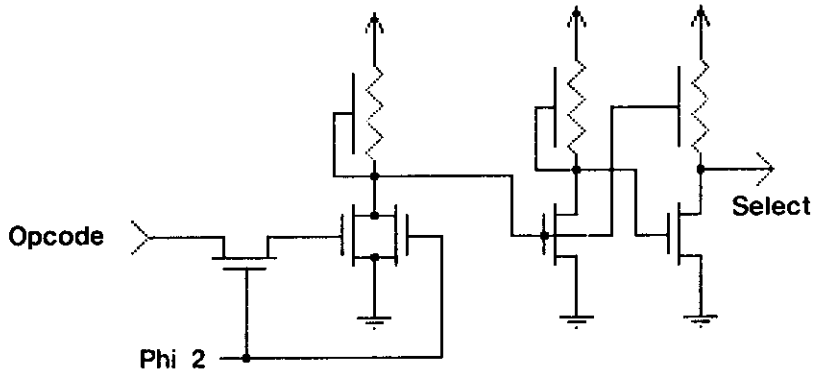


Figure 9. Select Control Driver.

All outputs low during Phi 2
(Precharge)
Selected terms high during Phi 1
Opcode valid during Phi 2

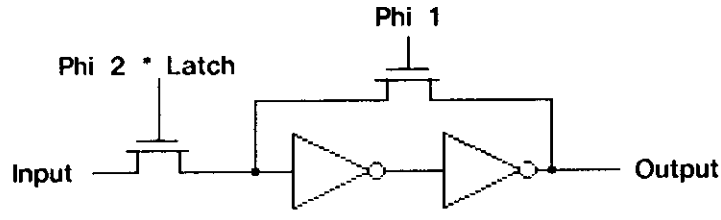


Figure 10. Output Register

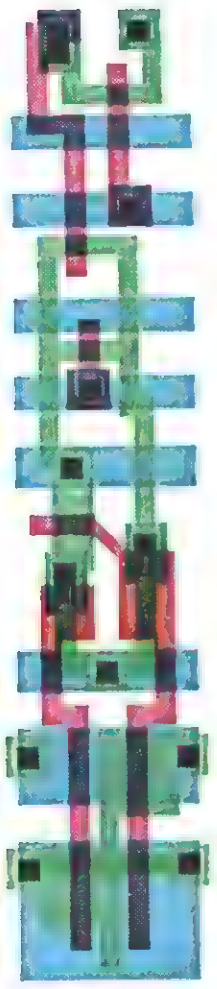


Figure 7a.

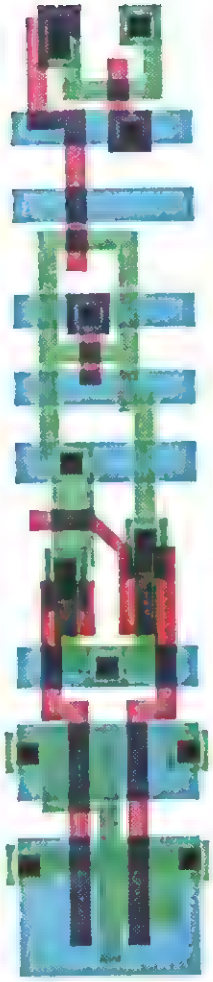


Figure 9a.

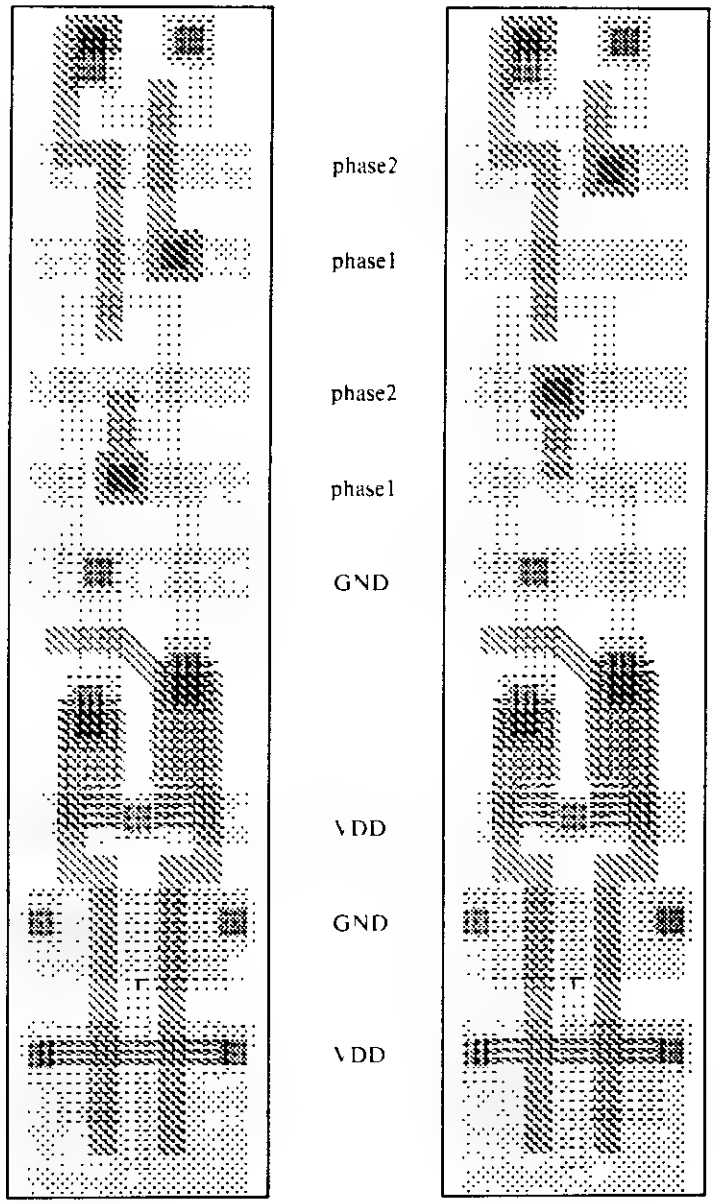


Fig. 7a.

Fig. 9a

to the particular P, K, or R term being driven. The only interface specification for the ALU which must be passed to the next level of system design is that the P, K, and R terms be valid before the end of φ_1, and that the A and B inputs likewise be valid by the end of φ_1 and be stable throughout φ_2, the active period of the ALU. We are then guaranteed that after enough time has passed to allow the carry to propagate, the output of the R function block will accurately reflect the specified function of the ALU and may be latched at the end of φ_2.

ALU Registers

In order for the arithmetic logic unit described in the last section to be useful, it must be equipped with a set of registers both for its input variables and for its output. Let us consider the input registers first. Inputs to the ALU may be derived from either the shifter, the buses, or other sources. They may be latched and left unchanged during any machine cycle or set of machine cycles. This is one of the situations in which combining the multiplexing function with the latching function simplifies the design and achieves better performance. A register operating in this manner is shown in Fig. 8.

The input to the first inverter can be derived from four sources: three internal sources such as shifter output, bus, etc., and a fourth, the output of the second inverter. When it is desired to latch a new signal into the register, one of the source pass transistors is driven high during φ_1. The feedback transistor around the two inverters is always activated during φ_2. Thus, with three vertical control wires plus the φ_2 timing signal, it is possible to select one of three sources into the register, or none of the three sources, thereby leaving the previous value of the register stored on the gate of the first inverter during the φ_1 period. Since it is necessary to have two inverters to form the stable pair when the feedback transistor is on, both the input and its complement are available as required by the P and K function blocks of the arithmetic logic unit. The OP code signal which selects which source will be applied to the ALU input register during φ_1 must come in during the previous φ_2. Each of the select signals must be low during φ_2, and at most one of them may come high during the following φ_1. A driver appropriate for these control signals is shown in Fig. 9. The control OP code is latched during φ_2, during which time the NOR gate shown disables the output driver. Since the output driver in this case is non-inverting, the output select line is held low during all of φ_2. At the end of φ_2, the OP code signal is latched and the particular select line to be enabled that cycle is allowed to go high.

Note that this timing allows two incoming OP code bits per external wire per machine cycle. In particular, if it were desirable to share a microcode bit between the ALU function and the ALU selector inputs, this could be done by bringing the ALU OP code in during φ_1 and the ALU input selection code in during φ_2, as shown in Fig. 10. This technique was suggested by Ivan Sutherland.

The ALU output register is similar to the ALU input register, except the timing is reversed. The result of the ALU operation is available at the end of φ_2.

An OP code bit will, if desired, enable the latch signal to go high during φ_2. The feedback transistor is always enabled during φ_1, and thus the latch is effectively static even though in the absence of a latching signal the data is stored dynamically on the gate of the first inverter through the φ_2 period. Once again, both the output and its complement are available if desired.

Buses

An early design decision was that data would flow through the machine on two buses which communicate with all of the major blocks of the system. We have already seen that the ALU performs its operation during the φ_2 period and does not have valid data to place into its output register until the end of φ_2. If data are to be transferred from the output register of the ALU to its input register, this must be done during the φ_1 period. If we adopt a standard timing scheme in which all transfers on the buses occur during φ_1, we can make use of the φ_2 period when the ALU is performing its operation to precharge the buses in the same manner that the carry chain was precharged during the φ_1 period. In this way we solve one of the knotty problems associated with a technology designed for ratio logic. If we had insisted that the tristate drivers associated with various sources of data for a bus be able to drive up as well as down, we would have required both a sourcing and sinking transistor, together with a method for disabling both transistors. While it is perfectly possible to build such a driver (we shall undertake the exercise as part of the design of the output ports), it is a space-consuming matter to use such a driver at every point where we wish to source data onto an internal bus. By using the bus precharge scheme, our tristate drivers become simply two series transistors as shown in Fig. 11.

Here the data from one source, for example the ALU output register, is placed on the gate of one of the series transistors. An enable signal which may come high during φ_1 is placed on the other series transistor. If one and only one of the enable signals is allowed to come high during any one φ_1 period, the bus can be driven from as many sources as necessary. The performance of such a bus is limited only by the pull-down capability of the two series transistors. We shall adopt this philosophy for the processor chip we are designing, and attach such a tristate driver to each of the output registers for the ALU.

Barrel Shifter

Since shifting is basically a simple multiplexing function, it might be thought that a shifter could be combined with the input multiplexer to the ALU. A simple 1-bit, right-left shifter implemented in this manner is shown in Fig. 12.

It is identical with the three-input ALU register, and the three inputs have been used to select between the bus, the bus shifted left by one, and the bus shifted right by one. To support the multibit shifts necessary for field extraction and building up odd bit arrays, something more is required. One is tempted initially to build up a multibit shift out of a number of single shifts. However, for word lengths of practical interest, the n^2 delay problem mentioned in Chapter 1 makes such an approach unworkable.

The basic topology of a multibit shift dictates that any bus bit be available at any output position. Therefore, data paths must run vertically at right angles to the normal bus data flow. Once this simple fact is squarely faced, a multibit shifter is seen as no more difficult than a single bit shifter. A fundamental circuit which allows any bit to be connected to any output position is shown in Fig. 13a. It is basically a crossbar switch with individual MOS transistors acting as the crossbar points. In principle this structure can be set to interchange bits as well as shift them, and is completely general in the way in which it can scramble output bits from any input position. In order to maintain this complete generality, the control of the crossbar switch requires n^2 control bits. In some applications, this n^2 bits may not be excessive, but for most applications a simple shift would be adequate. The gate connections necessary to perform a simple barrel shift are shown in Fig. 13b. The shift constant labelled SC, is presented on n wires, one and only one of which is high during the period the shift is occurring. If the shift outputs, SH0,1,2,etc., are precharged in the same manner as the bus, the pass transistors forming the shift array are only required to pull

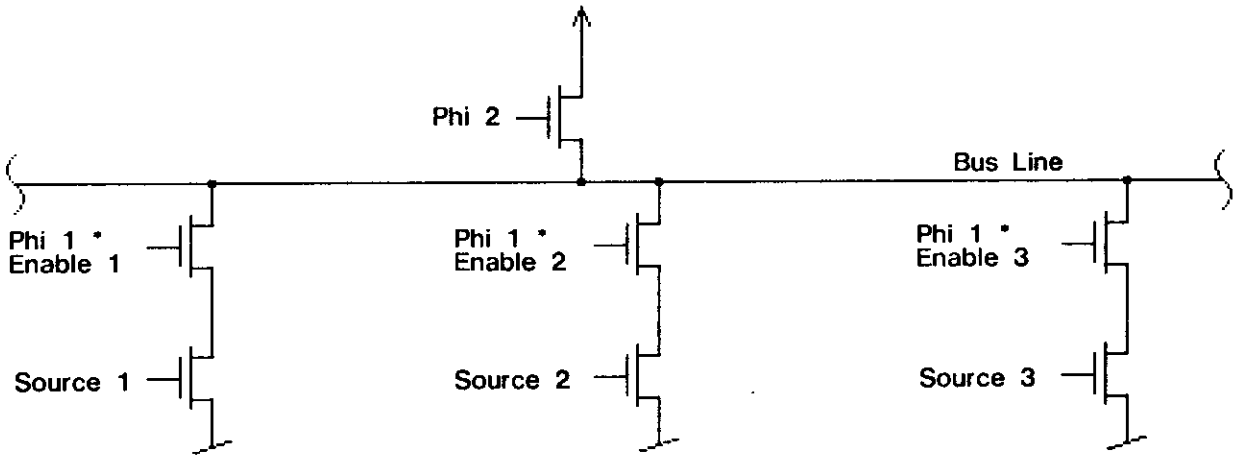


Figure 11. Precharged Bus Circuit.

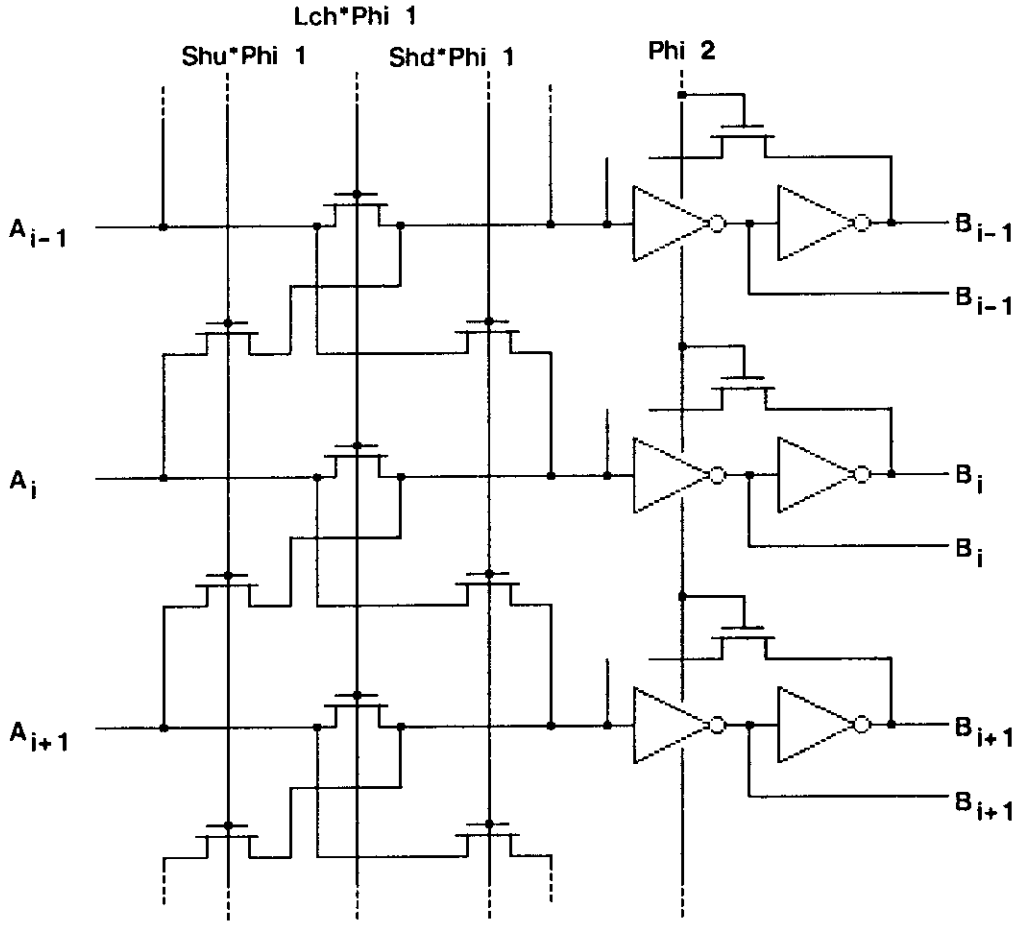


Figure 12. A Simple 1-Bit, Right-Left Shifter.

down the shift outputs when the appropriate bus is pulled to low by its tristate drivers. Thus, the delay through the entire shift network is minimized and effective use is made of the technology.

A second topological observation is that in every computing machine, it is necessary to introduce literals from the control path into the data path. However, our data path has been designed in such a way that the data bits flow horizontally while the control bits from the program store flow vertically. In order to introduce literals, some connection between the horizontal and vertical flow must occur. It is immediately obvious in Fig. 13 that the bus is available running vertically through the shift array. It is then the obvious place to introduce literals into the data path or to return values from the data path to the controller.

At the next higher level of system architecture, the shift array bit slice may be viewed as a system element with horizontal paths consisting of the bus, the shifter output, and if necessary, the shift constant since it appears at both edges of the array, as well. The literal port is available into or out of the top edge of the bit slice, and the shift constant is available at the bottom of the bit slice. These slices, of course, are stacked to form the entire shift array as wide as the word of the machine being built.

One more observation concerning the multibit shifter is in order. We stated earlier that our machine was to be a 2-bus machine. Therefore, any bit slice of a shifter such as the one shown in Fig. 13 will of necessity have two buses running through it rather than one. We chose to show only one for the sake of simplicity. There remains the question of how the two buses are to be integrated with the shifter. Since we are constructing a two-bus machine, we have two full words available, and a good field extraction shifter would allow us to extract a word which gracefully crosses the boundary between two machine words. The arrrangement shown in Fig. 13 performs a barrel shift on the word formed by one bus. For the same number of control lines and pass transistors, only having added the bus lines which are required for the balance of the machine anyway, we may construct a shifter which places the words formed by the two buses end to end and extracts a full-width word which is continuous across the word boundary between the A and B buses. This function is accomplished in as compact a form as just described with a circuit shown in Fig. 14. Notice that the vertical wires have a split in them. The portion of the wire above the corresponding shift output being connected to the A bus, and that below the corresponding shift output to the B bus.

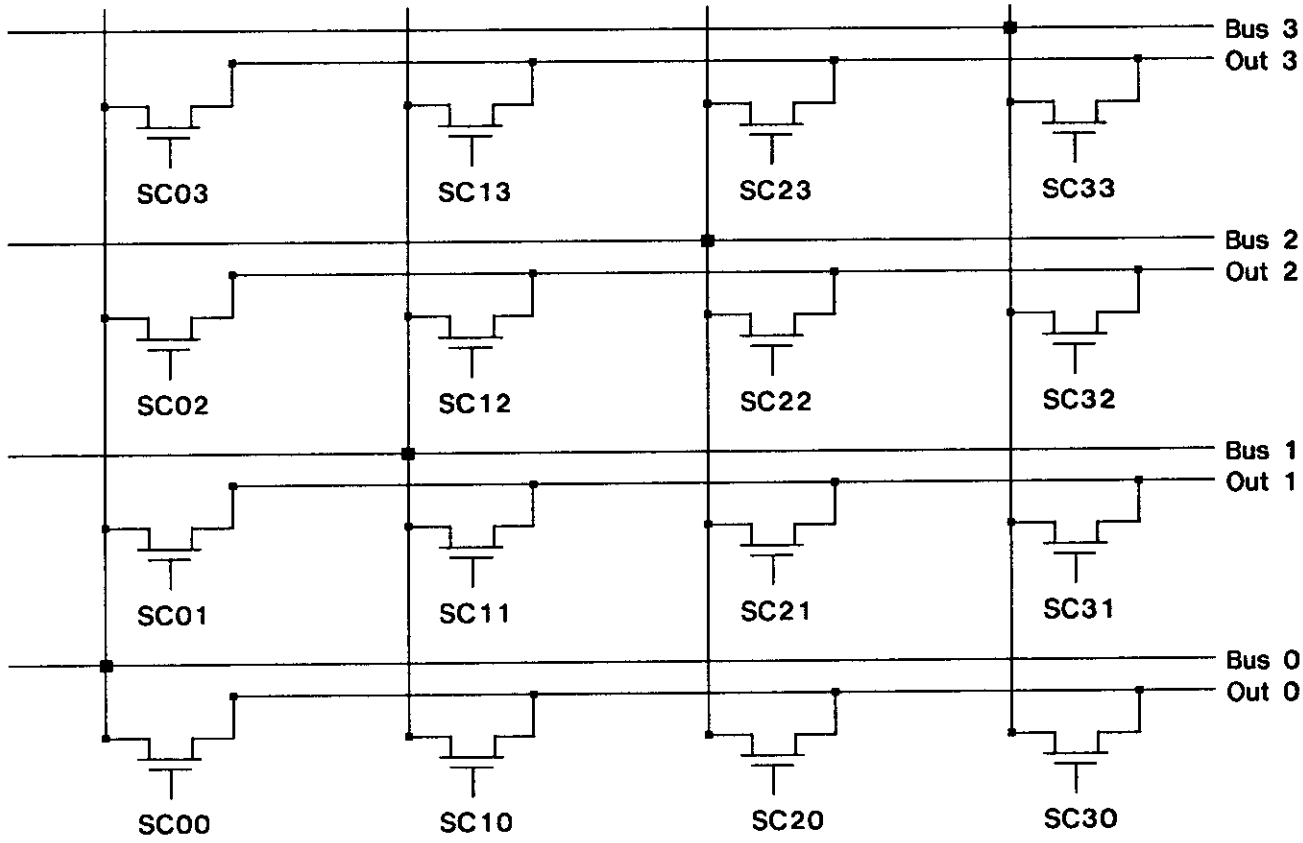


Figure 13a. 4-By-4 Crossbar Switch.

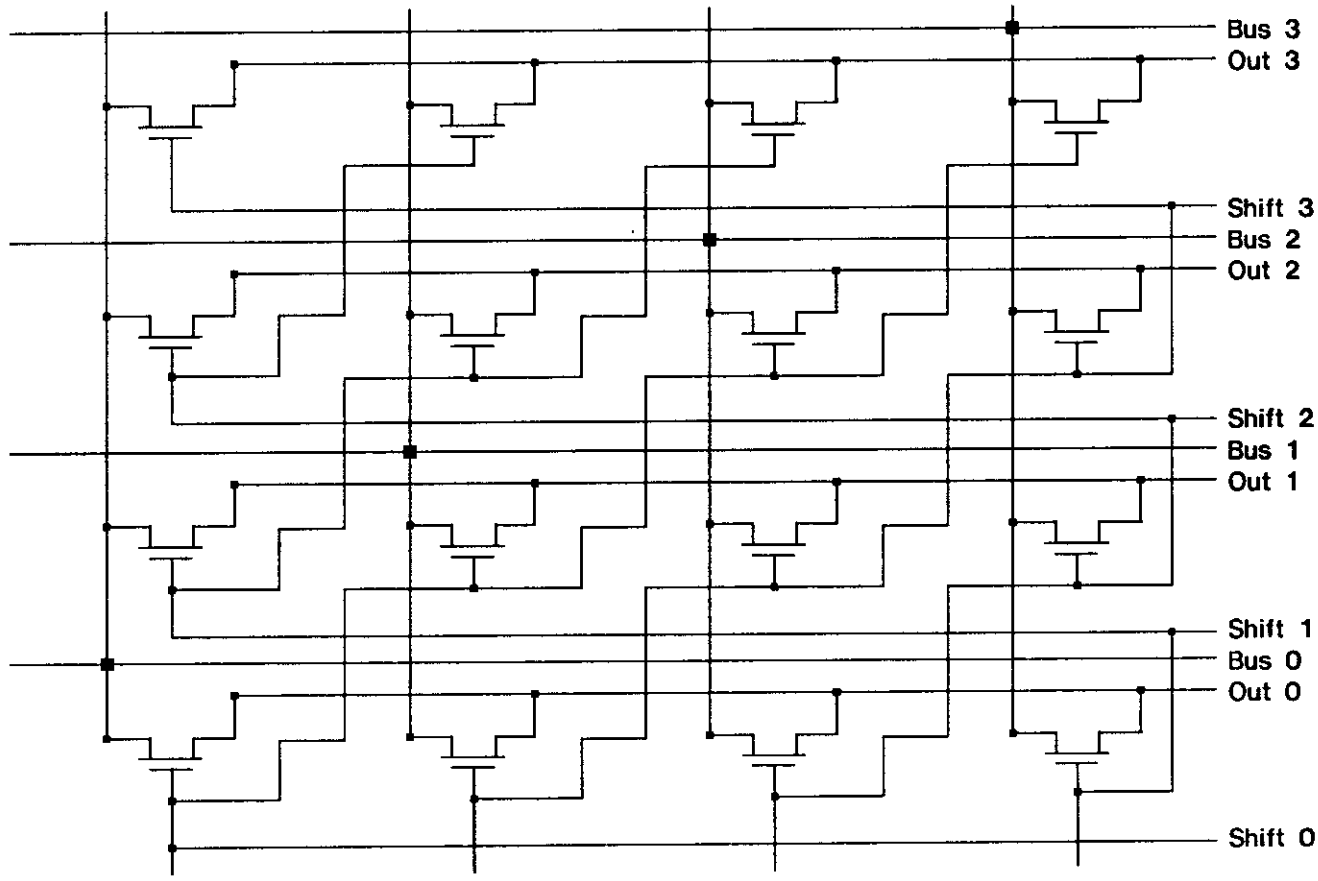


Figure 13b. 4-By-4 Barrel Shifter.

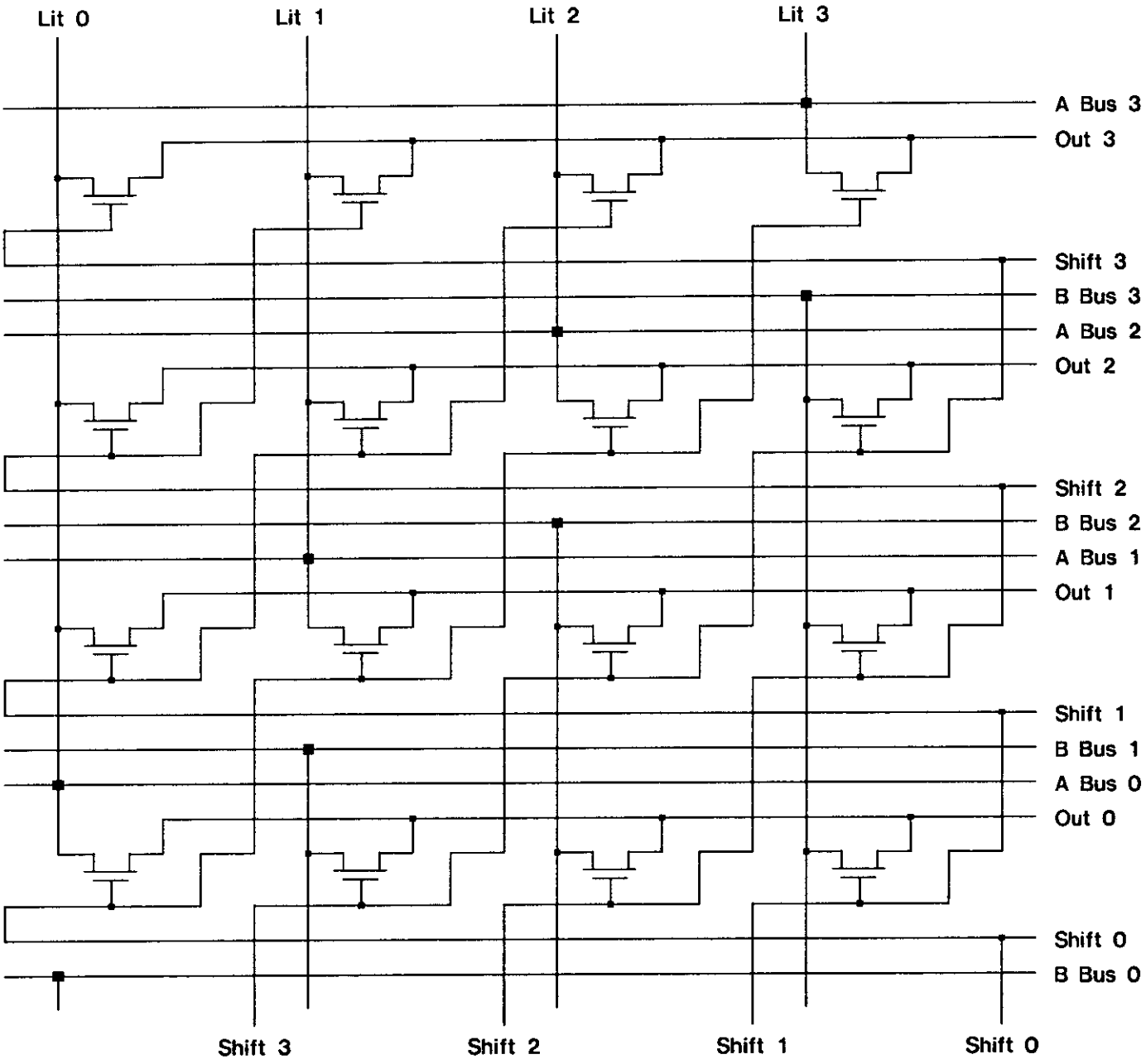


Figure 14. 4-By-4 Shifter with Split Vertical Wires and 2 Data Buses.

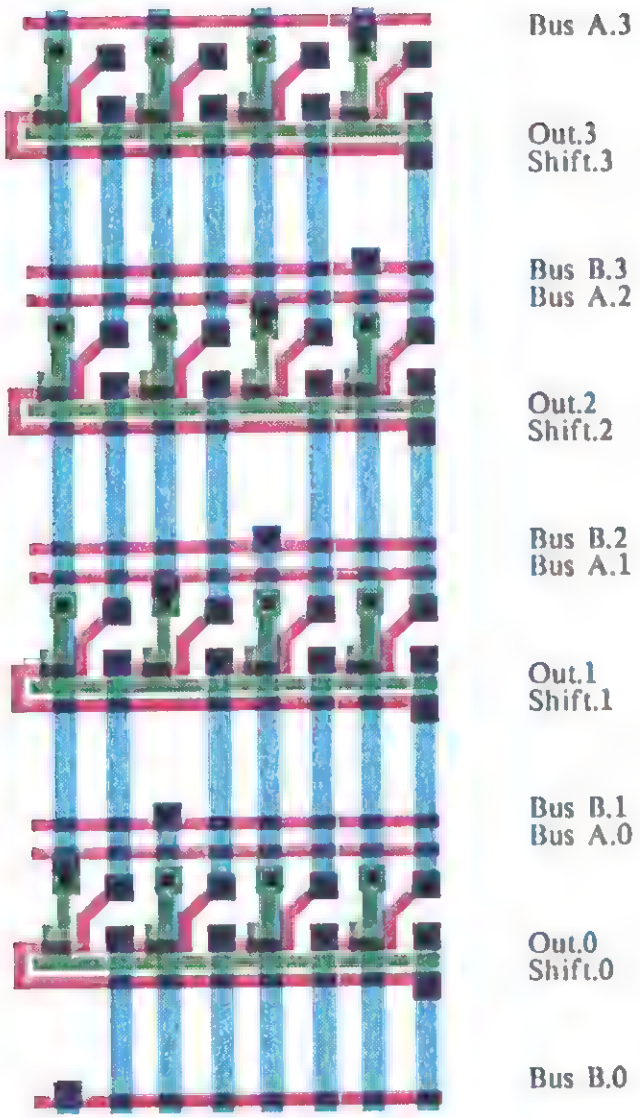

Bus A.3
Out.3
Shift.3
Bus B.3
Bus A.2
Out.2
Shift.2
Bus B.2
Bus A.1
Out.1
Shift.1
Bus B.1
Bus A.0
Out.0
Shift.0
Bus B.0

Figure 14a.

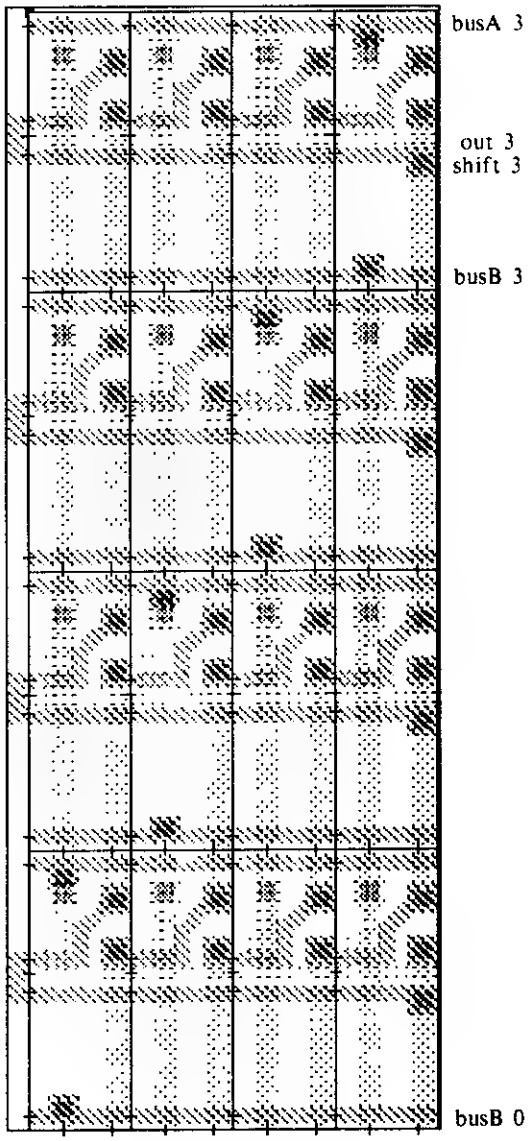


Fig. 14a.

It can be seen by inspection that this circuit performs the function shown in Fig. 15 which is just what is required for doing field extractions and variable word length manipulations. The literal port is connected directly to the A bus and may be run backwards in order to discharge the bus when a literal is brought in from the control port. A block diagram which represents the shifter at the next level of abstraction is shown in Fig. 16.

In order to complete the shifter functional block, it is necessary to define the drivers on the top and bottom which interface with the system at the next higher level. Let us assume that the literal bus from outside the chip will contain data which are valid on the opposite phase of the clock from that of the internal buses. In that case, a very simple interface between the two buses which will operate in either direction is shown in Fig. 17.

The internal shifter output is precharged during φ_2, and active during φ_1. It may be sourced either from the literal bus or from the shifted combination of the A and B buses through the shift array, shown in Fig. 15. The external literal bus itself may be sourced either from the opposite end (the external paths from the program source) or from the end attached to the A-Bus in the shift array shown.

The bus to the external literal path is precharged during φ_1, and data from the literal port of the shifter are enabled onto it by a signal active during φ_2, as shown in Fig. 17. The two signals, φ_1 * IN, and φ_2 * OUT, are derived from buffers identical to those shown earlier. The shift constant itself is represented by one line out of n, which is high, the others remaining low. Buffers for these lines are identical to those shown in Fig. 9.

There is one more observation concerning the n-bit shift constant. It is represented most compactly by a log n bit binary number. However, in order to generate from such a form a signal that can be used in the actual data path, a decoder is required to convert the binary number into a one-of-n signal suitable for feeding the buffers. Decoders can be made in a number of ways in the ratio technology we are discussing. The most common form is the NOR form, which is merely the fully decoded equivalent of the AND-plane in the programmable logic array, Chapter 3. It is shown in Fig. 18. Notice that the output is a high-going one-of-n pattern.

Decoders can also be made in other forms. For small values of n, the NAND form shown in Fig. 19 is often convenient. We used a variant of this form for the ALU function block described earlier. Notice that the output of this form, when used as a decoder, is a lowgoing

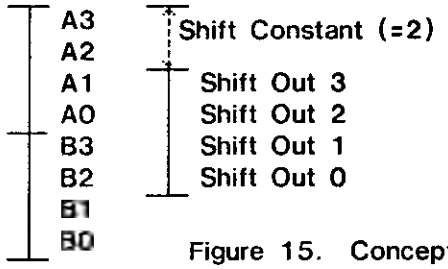


Figure 15. Conceptual Picture of the Shifter's Operation.

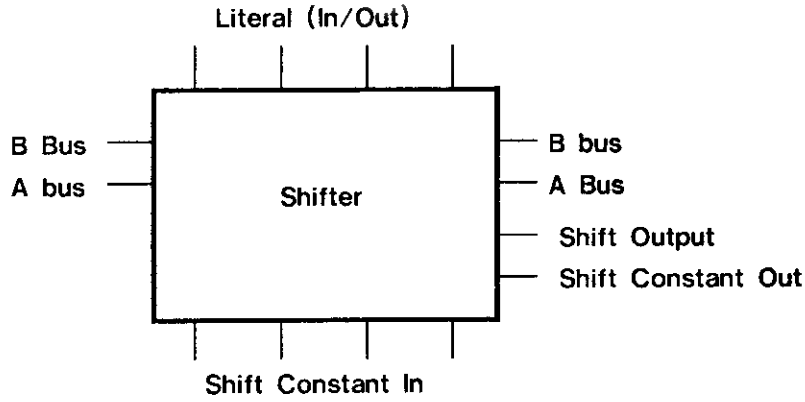


Figure 16. Block Diagram of the Shifter.

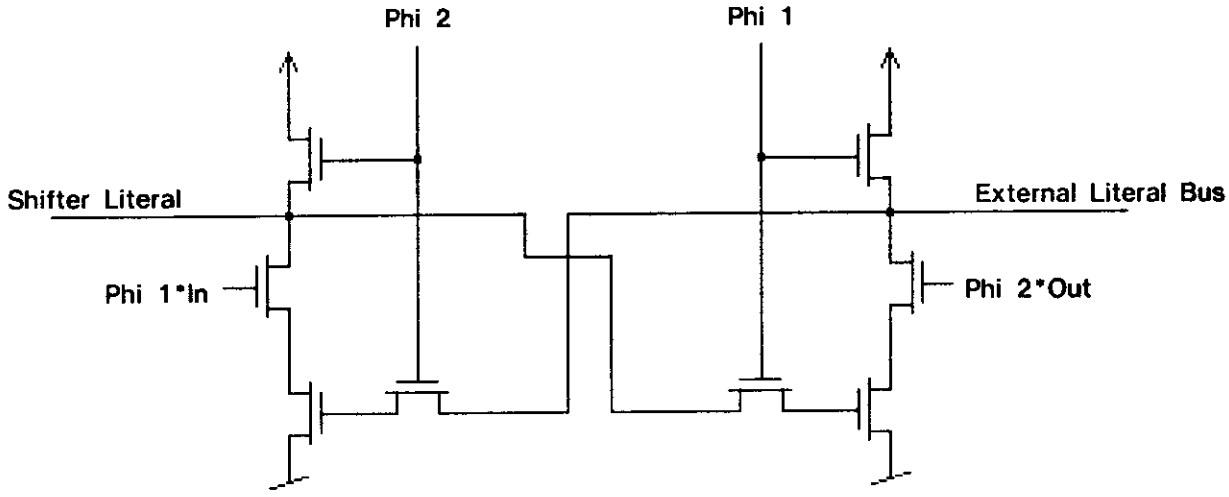


Figure 17. Literal Interface.

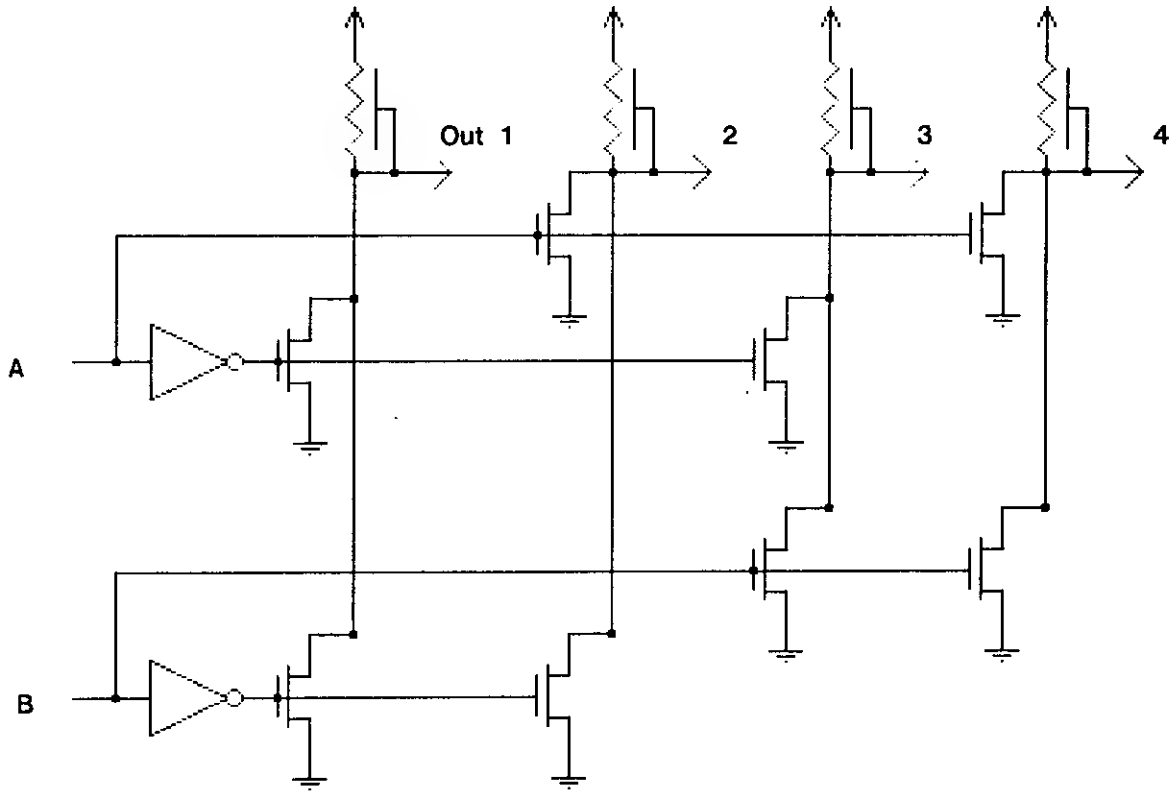


Figure 18. A Nor Form 1-of-N Decoder.

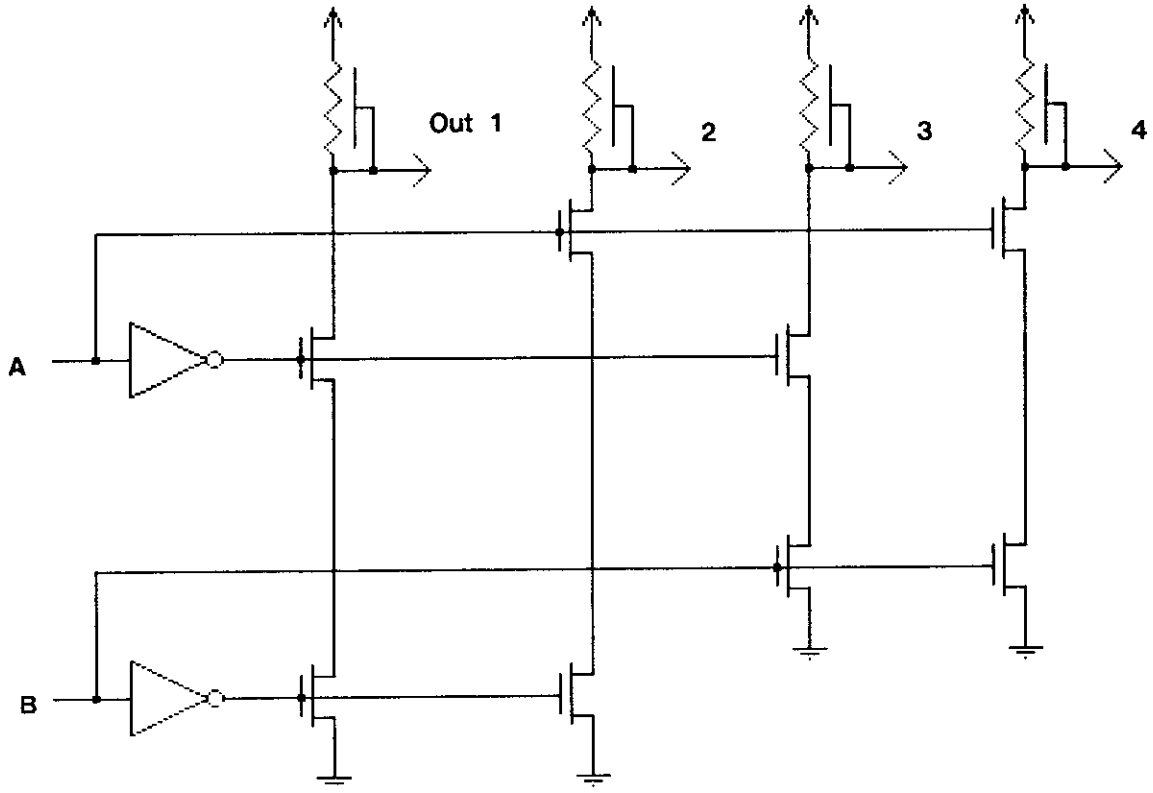


Figure 19. A Nand Form 1-of-N Decoder.

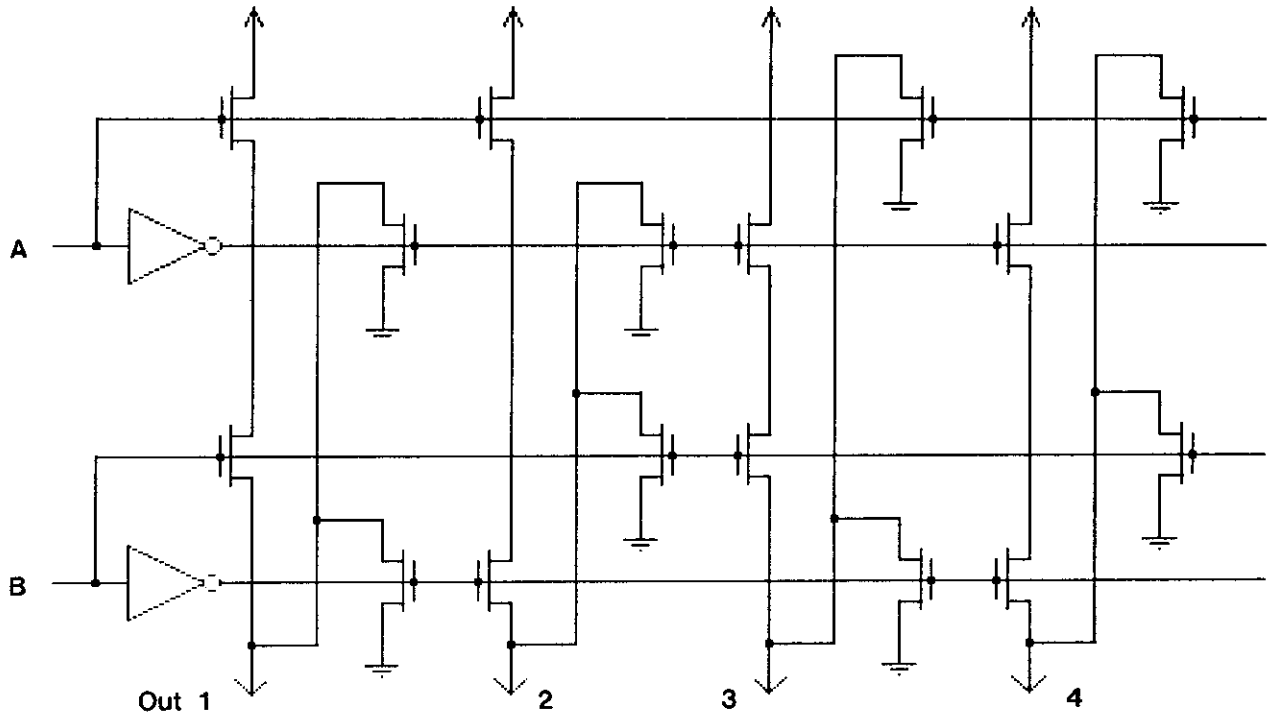


Figure 20. A Complementary Form 1-of-N Decoder.

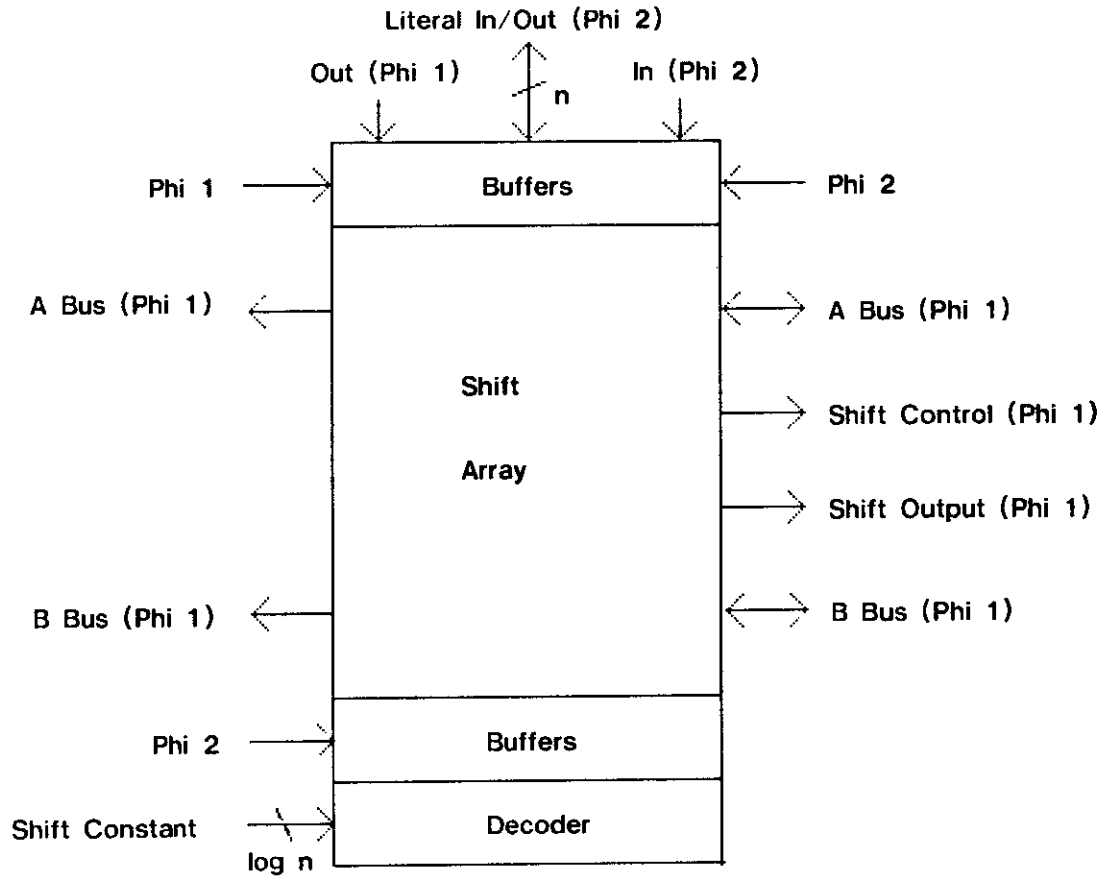


Figure 21. A Fully Synchronized Shifter.

one-of-n pattern. There is also a complementary form of decoder which can be built with ratio technology, and was suggested by Ivan Sutherland. It takes advantage of the fact that in any decoder both the input term and its complement must be present. In this case, the input term can be used to activate pull-up transistors in series, while the complement can be used to activate pull-down transistors in parallel. This logic form is similar in principle to that used with fully complementary technologies, and has similar benefits. It can generate either a highgoing or a lowgoing one-of-n number, and dissipates no static power. A decoder of this sort is shown in Fig. 20. Once we have added the appropriate buffers and decoders to our shift array, we have a fully synchronized function block ready to be integrated with the system at the next level up. The properties of this block are shown in Fig. 21.

Register Array

In any microcoded machine designed for emulating an instruction set at a higher level, it is convenient to have a number of miscellaneous registers available, both for working storage during computations and for storing pointers of specific significance in the machine being emulated: stack pointers, base registers, program counters, etc. Since the machine is a two-bus machine and the ALU is a two-operand device, it is convenient if the registers in our machine are two-port registers. Using the design philosophy we have been discussing, a typical two-port register cell is shown in Fig. 22. This register is a simple combination of the input multiplexer described earlier, the φ_2 fedback transistor, and two tristate output drivers, one for each bus. The registers can be combined into an array m bits long and n bits wide, the buses passing through the array. Each cell of the array is defined at the next level up, as shown in Fig. 23. Drivers for the load inputs and the read outputs are identical to those shown in Fig. 9. While we could immediately encode the load and read inputs to the registers into log n bits, we shall delay doing so until the next level of system design. There are a number of sources for the A bus besides the registers, and we will conserve microcode bits by encoding them together.

Before we proceed, there is one mundane detail which must be taken care of in the overall topological strategy. The routing of VDD and Ground must generally be done in metal, except for the very last runs within the cells themselves. Often the metal must be quite wide, since metal migration tends to shorten the life of conductors if they operate at current densities much in excess of 1 ma per square micron cross-section. Thus, it is important to

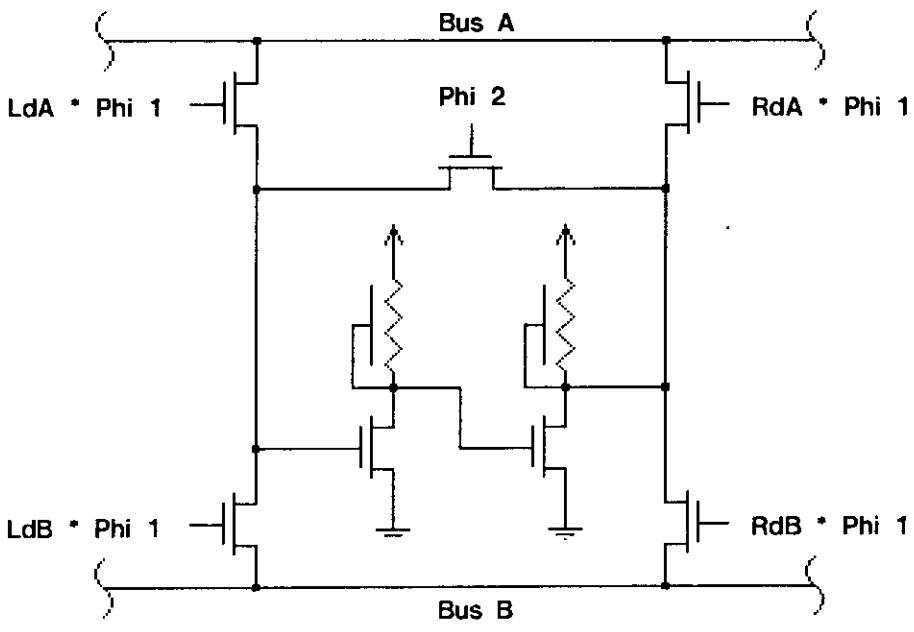


Figure 22. A Two Port Register Cell.

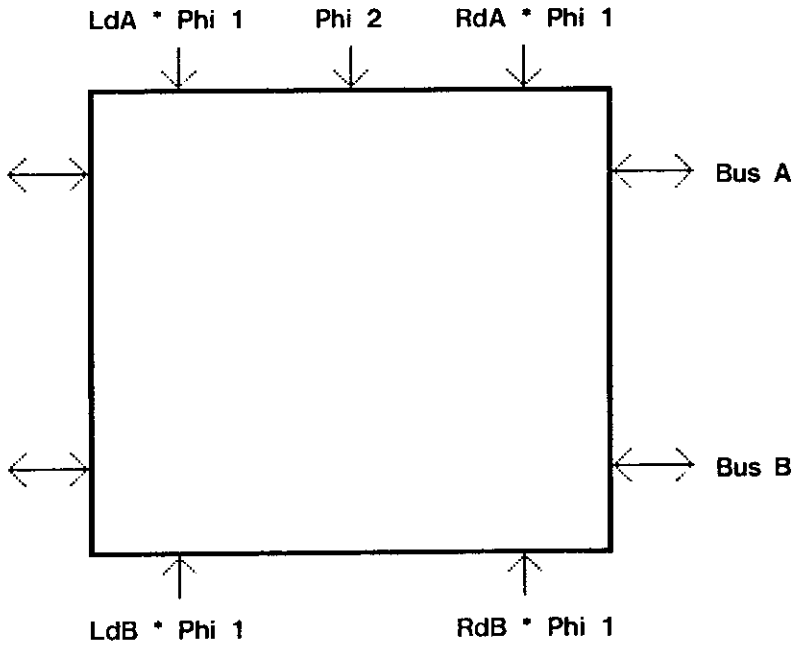


Figure 23. Block Diagram Definition of the Two Port Register Cell.

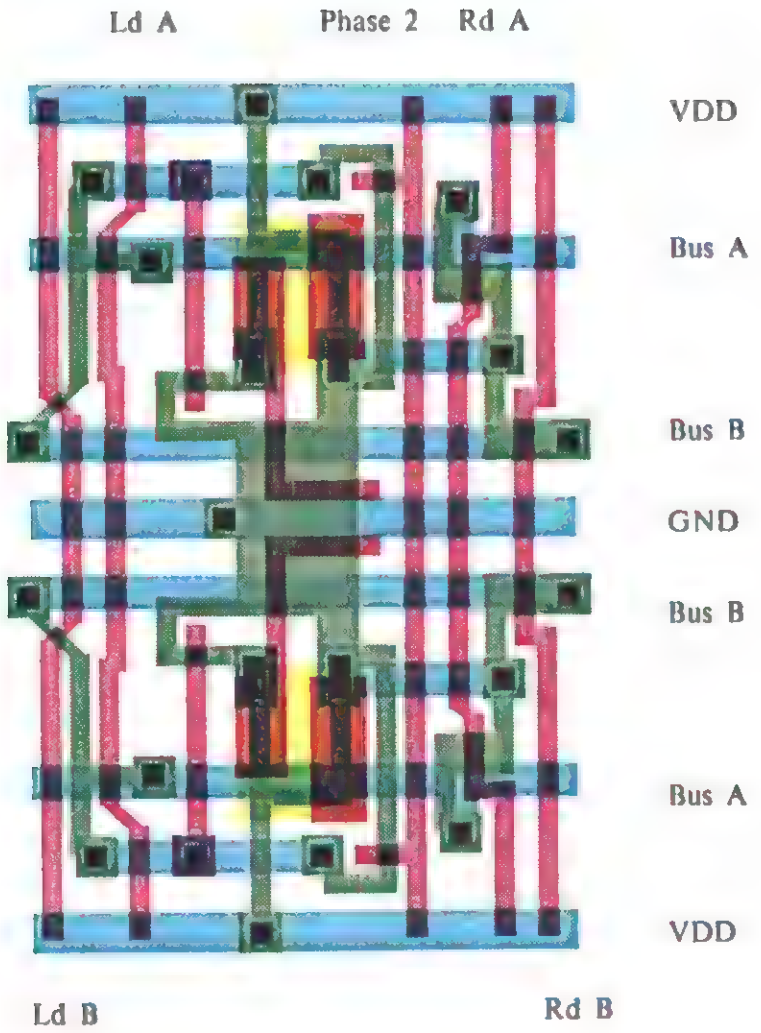


Figure 22a.

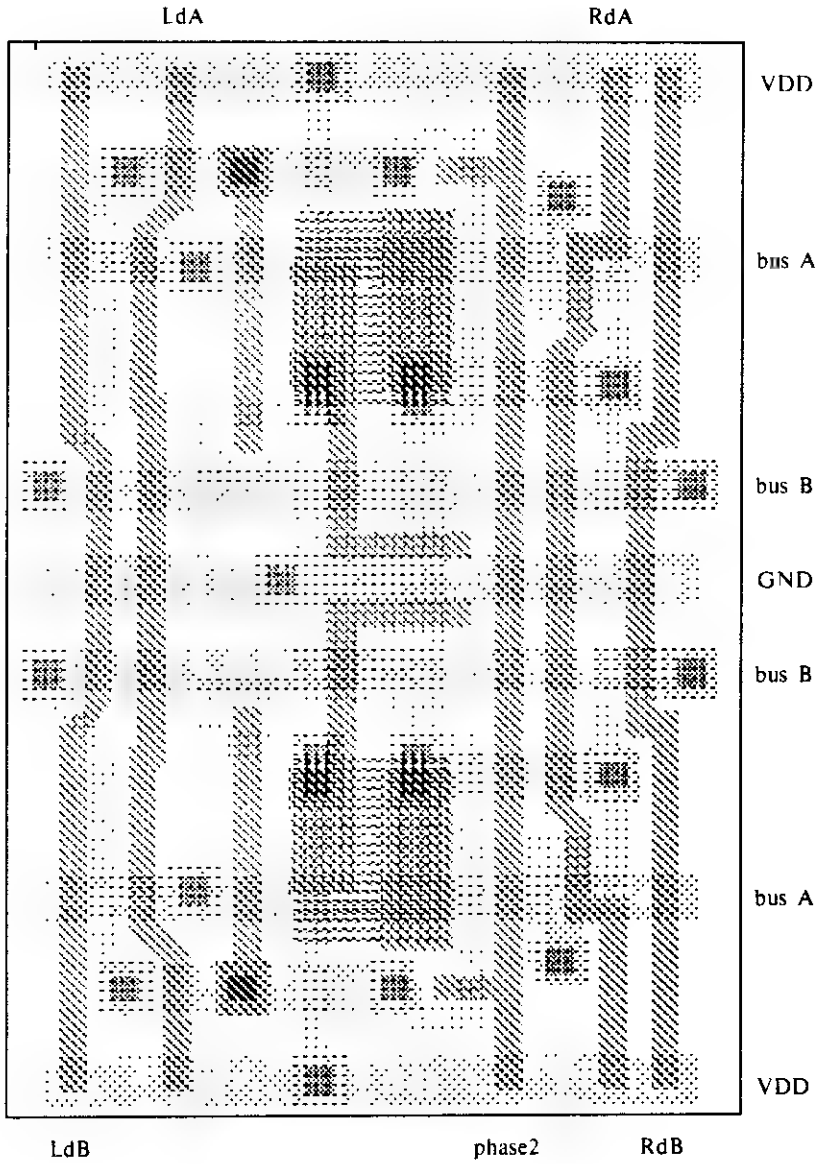


Fig. 22a.

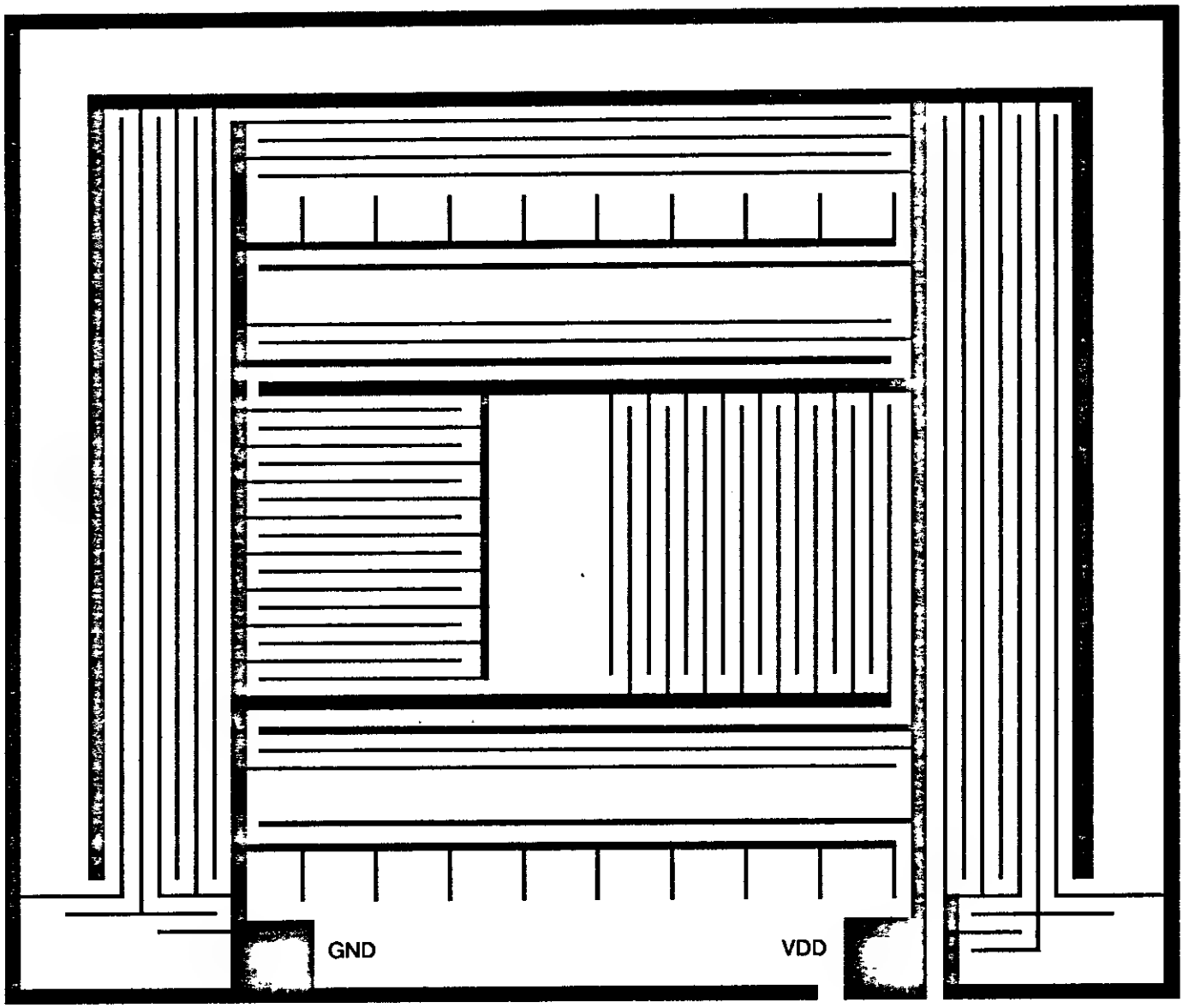


Figure 24. VDD and GND Net for the Data Processing Structure Shown in Figure 1.

have a strategy for routing ground and VDD to all the cells in the chip before doing the detailed layout of any of the major functional blocks. Otherwise, one is apt to be faced with topological impossibilities because certain conductors placed for other reasons interfere with the routing of the VDD and ground. A possible strategy for the overall chip layout shown in Fig. 1 is shown in Fig. 24.

Notice that the VDD and ground paths form a set of interdigitated combs, so that both conductors can be run to any cell in the chip. Any strategy will do, but it must be consistent, thoroughly thought through at the beginning, and rigidly adhered to during the execution of the project.

Communication with the Outside World

Although in particular applications the interface from a port of the machine to the outside world may be a point to point communication, the ports will often connect to a bus. Thus it is desirable to use port drivers which may be set in a high impedance state. Drivers which can either drive the output high, drive the output low, or appear as a high impedance to the output are known as *tristate* drivers. Such drivers allow as many potential senders on the bus as necessary. Figure 25 shows the circuit for a tristate interface to a contact pad.

Here, either bus A or bus B can be latched into the input of a tristate driver during φ_1. Likewise the pad may be latched into an incoming register at any time independent of the clocking of the chip. Standard tristate drivers are enabled on bus A and B. The only remaining chore is the design of the tristated buffer which drives the pad directly. Details of the tristate driver are shown in Fig. 26.

The terms out and outbar are fed to a series of buffer stages which provide both true and complement signals as their outputs, and are disabled by a DISABLE signal. Note that this DISABLE signal does not cause all current to cease flowing in the drivers, since the pull-up transistors are depletion type, but reduces the current to a value where it can be handled by the disable transistor of the following buffer stage. In general there will be a number of super buffer stages of this sort. The very last stage of the driver is shown in Fig. 26b. It is not a super buffer but employs enhancement mode transistors for both pull-up and pull-down. These transistors are very large in order to drive the large external capacitance associated with the wiring attached to the pad. They are disabled in the same manner as the

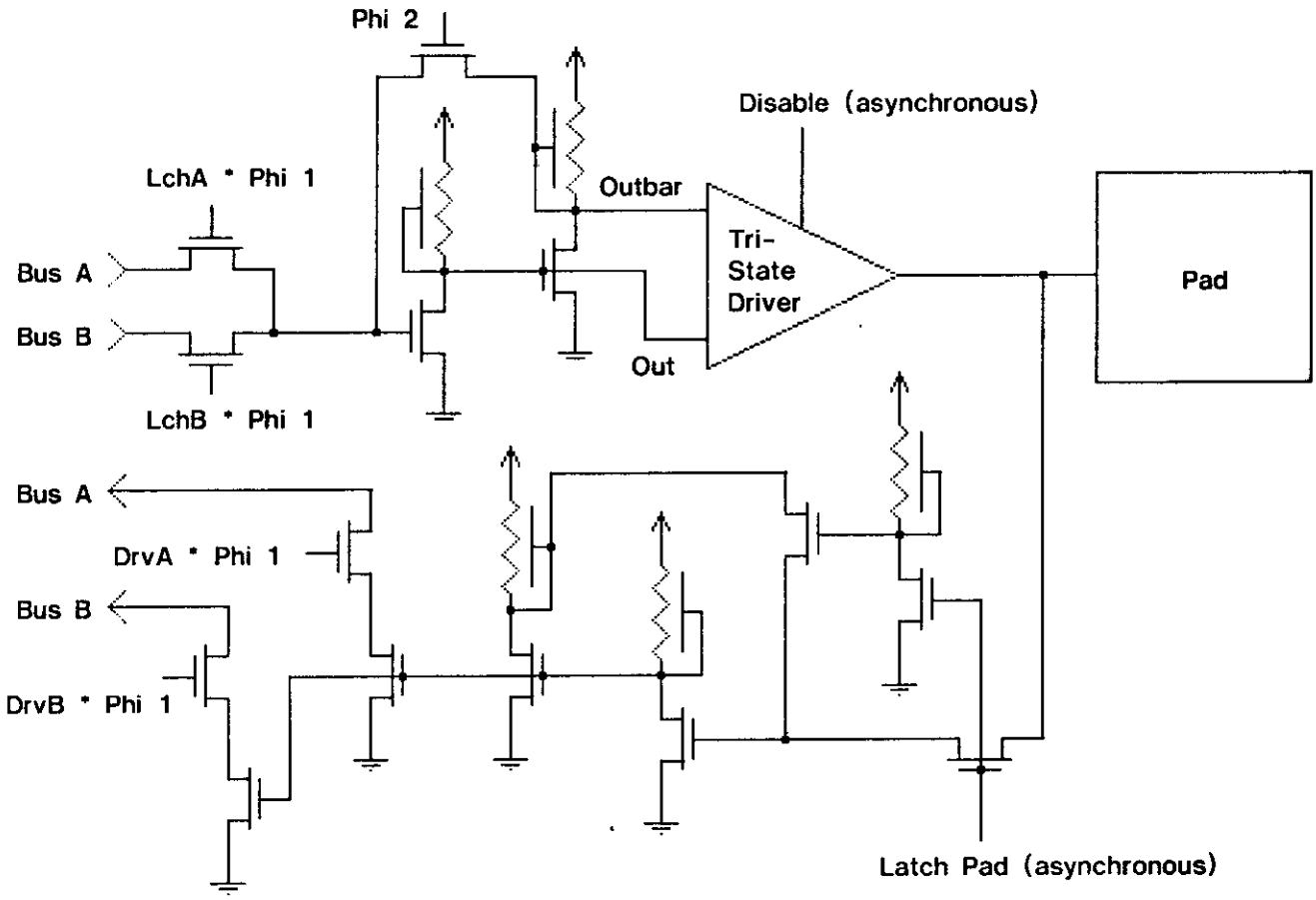


Figure 25. Data Port Tristate Pad Circuit

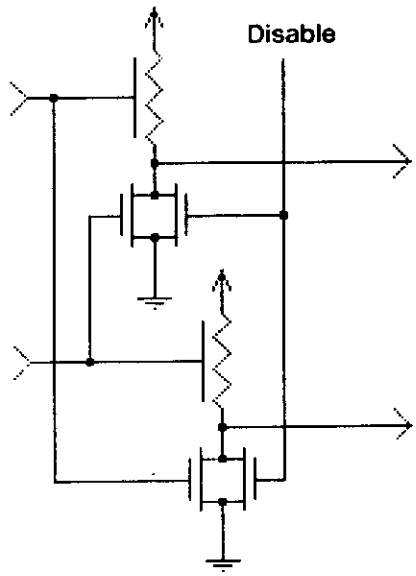


Figure 26a. Pad Buffer Stage.

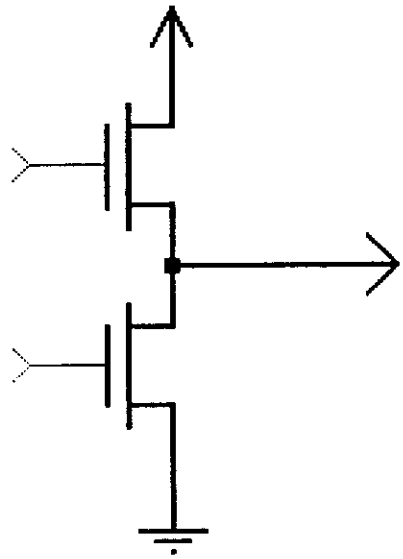


Figure 26b. Pad Output Driver.

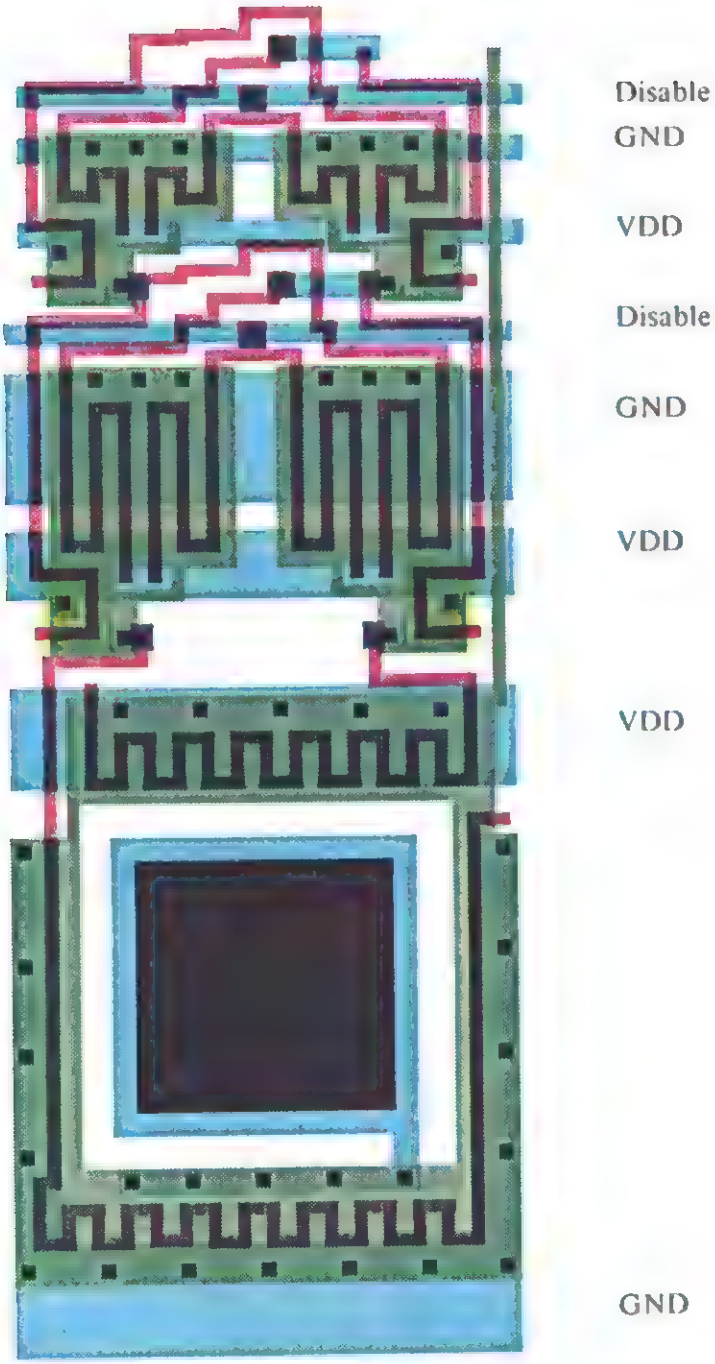


Figure 25a.

super buffers, except that when the gates of both transistors are low, the output pad is truly tristated. Once again the two output transistors are a factor of approximately e larger than the last super buffer in the buffer string.

As we have seen, the inverter string necessary to transform the impedance from that of the internal circuits on chip to that sufficient for driving a pad attached to wiring in the outside world is quite large, and imposes a delay of some factor times a logarithm of this impedance ratio upon communications between the chip and the outside world. Any help which can be obtained in making this transformation is of great value. For example, the latch and buffers associated with the input bus circuit to the pad drivers can themselves be graded in impedance level, so that by the time the out and outbar signals are derived, they are at a considerably higher current drive capability than the buses. Note that the buses are a considerably larger capacitance than minimum nodes on the chip, and thus the initial latch buffers can be larger than typical inverters on the chip. All such tricks help to minimize the number of stages between the bus and the outside pad, and thus the total delay in going off chip.

Machine Operation Encoding

By now we have defined a complete functional data path with ports on each end and functional blocks through the center, as shown in Fig. 27. The op code bits required to control the data path and the phase of the clock on which they are latched are shown. There are forty-nine such bits together with the four asynchronous bits for latching and driving the pad to the external world. In addition, there are the carry-out wire and the sixteen literal wires. These sixty-six wires together with the thirty-two from the left and right port must go to and come from somewhere. Schemes for encoding internal machine operations into OP codes of various lengths are well known, and will not be discussed here. At one extreme all the OP code wires can be brought out to a microcode memory driven by a micro program counter and controller, in which case all operations which can be done by the machine may be done in parallel. The opposite extreme is to very tightly encode the operations of the machine into a predefined OP code set. In the present machine, this encoding would be most conveniently done by placing a programmable logic array or set of programmable logic arrays along the top and the bottom of the machine data path. A condensed OP code could then be fed to the programmable logic arrays which in turn sequence the data path through the microinstructions required for executing machine code.

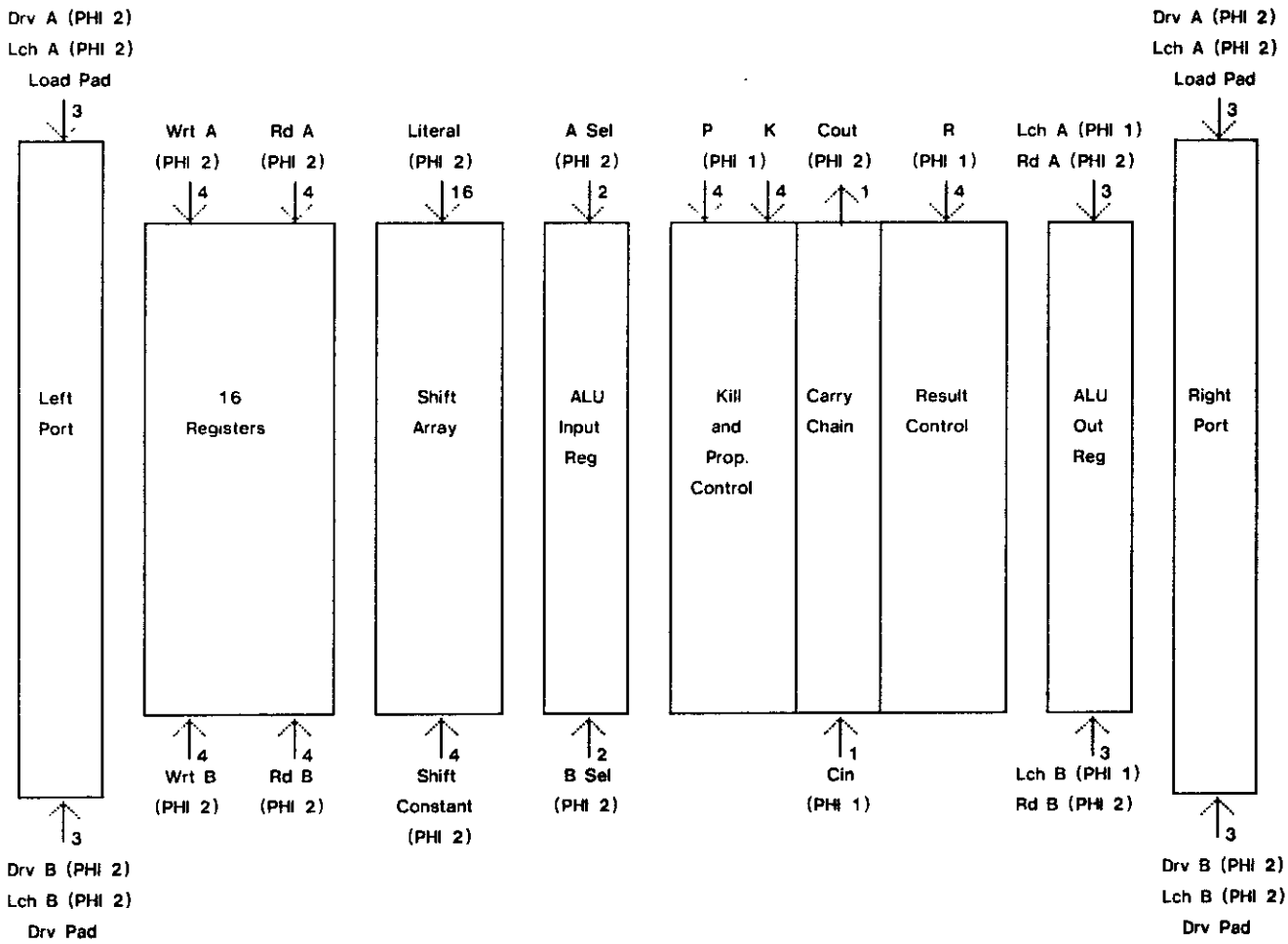


Figure 27. Block Diagram of Datapath with Control Wires Added.

The important point of the design strategy we have chosen is that we can orthogonalize the design of the data path and the design of the OP code set in such a way that the interface between the two designs is very well defined, very clean, and can be described precisely, in a way that system designers at the next higher level can understand and work with comfortably. The data path can then be viewed as a component in the next level system design. As time progresses and it is possible to construct chips with larger and larger functional density, blocks of the sort shown will form components in even larger geometrical arrangements which will form even larger components and a whole hierarchy will emerge which will implement a system function at a much higher level than contemplated here. However, if the design strategy we have described is followed, it is possible to construct arbitrarily large and complex systems which are guaranteed to work if the individual component blocks are correct, and given the clocking period is sufficient to allow the slowest functional unit to perform its function.

Using the approximate capacitance values given at the end of Chapter 2, an estimate of the minimum clock period for the machine can be made. The Phase 1 time of the machine is ~50τ, the same as the general estimate given in the section "Transit Times and Clock Periods" in chapter 1. However, the Phase 2 time of the machine is limited by the carry chain, as discussed earlier in this chapter. The relative areas of metal, diffusion, and gate can be estimated from the ALU layout shown in Figure 6a. The metal and diffusion occupy ~15 and ~8 times the area of the propagate pass transistor gate, respectively. Metal is ~0.1 and diffusion is typically 0.2 times the gate capacitance per unit area. Thus the total capacitance of each stage of the carry chain is ~4.5 times that of the pass transistor gate. The effective delay time is correspondingly longer than the transit time τ of the transistor itself. The effective delay through n stages of such pass transistor logic is ~ τn^2. In the OM2, n=4 and the effective delay for 4 bits of carry chain is ~4.5*16τ = 72τ. To this must be added the delay of the doubly inverting buffers at the end of every 4 bits of straight Manchester logic. This delay is (1+k) times the transit time of the inverter pulldown, properly corrected for stray capacitance in the inverter. Here the inverter ratio k is ~ 8, since its input is driven through the pass transistors. Conservatively, strays in such a circuit are always several times greater than the basic gate capacitance, and we may estimate the inverter delays at ~30τ. The total carry time is thus ~100 times the transit time for each block of 4 ALU stages. The total Phase 2 time should then be ~400τ. In 1978, τ ~ 0.3 ns, and we would expect a minimum total clock period of ~450τ, or ~135 ns.

The Second Half of this Chapter contains a functional specification of the OM2 machine, by Dave Johannsen of Caltech. This specification was originally documented in Display File #1111, by Dave Johannsen and Carver Mead of the Caltech Computer Science Department, and copyrighted by Caltech. The specification is reprinted here with the permission of the California Institute of Technology.

Functional Specification of the OM2 Machine

[Section contributed by David F. Johannsen, Caltech]

Introduction

This specification describes a 16-bit data engine referred to as OM2 [#986]. The OM2 contains 16 registers, an ALU, and a 32-bit shifter, and is designed as part of a micro-programmed writeable-control-store machine. The companion chip is the Controller chip, which contains the program counter, stacks, and so on. The Controller is described in Chapter 6. The entire system is designed to run on a single 5 volt supply.

The OM2 Datachip has two data ports for communication with the external system and a communication path to the Controller chip. The data ports are tri-state with either internal or external control. Communication with the Controller consists of a 16-bit literal port and a single flag bit. Seven control bits come directly from the microcode memory.

The system runs on a single clock. When the clock is high, the internal buses transfer data: when the clock is low, the ALU is performing its operation. Microcode bits enter the Datachip the phase before that code is to be executed. Therefore, the bus transfer code enters the Datachip when the clock is low, and the ALU code enters when the clock is high. Figure 1 shows a possible OM system.

Throughout this section a positive logic convention is used. A "1" refers to a high voltage level, while a "0" refers to a low voltage level.

Datapaths

A block diagram of OM2 is shown in figure 2. There are two buses which connect the various elements of the chip together. These buses transfer data while the clock is high, the period referred to as $\varphi 1$. During $\varphi 2$, when the clock is low, the buses are precharged. Each bus can only get data from one source, and give data to one destination during any one cycle.

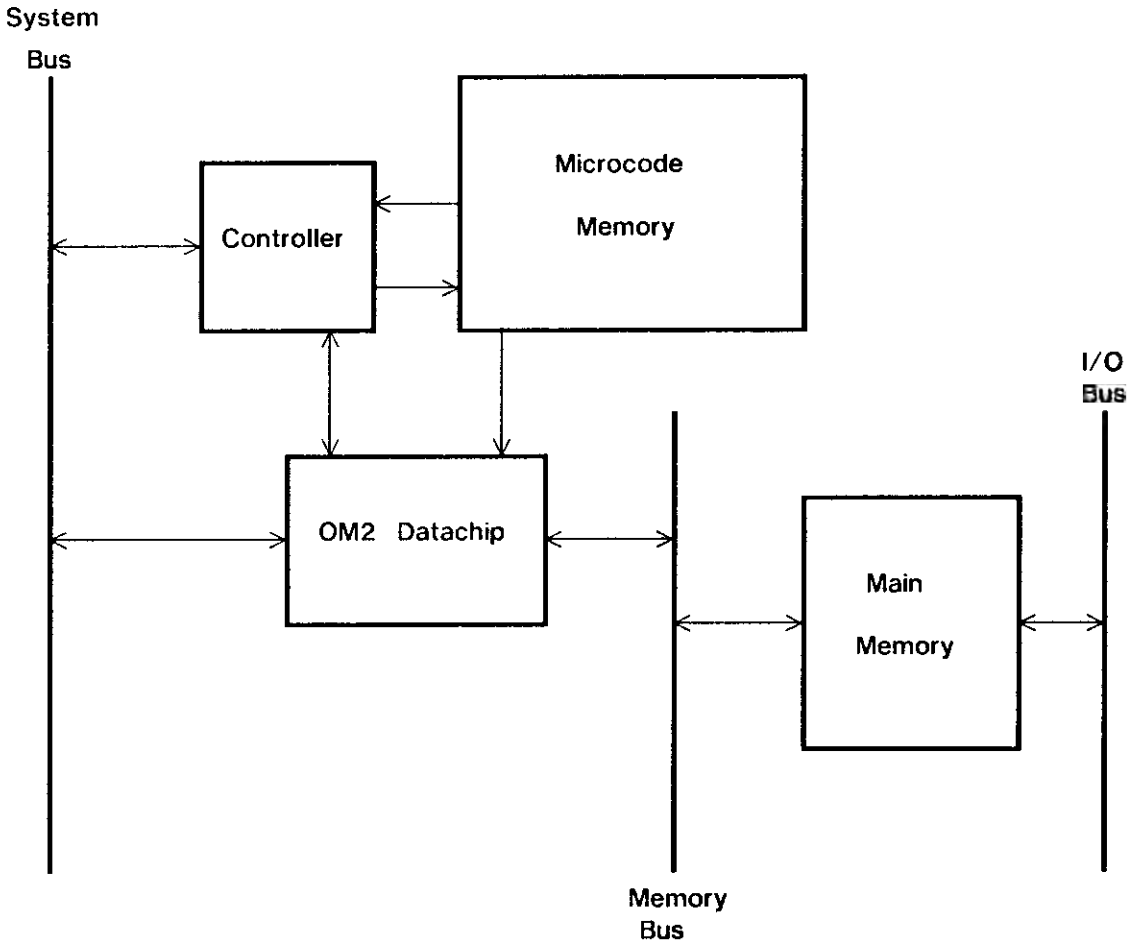


Figure 1. One Possible OM2 System Configuration

(fig1 sil)

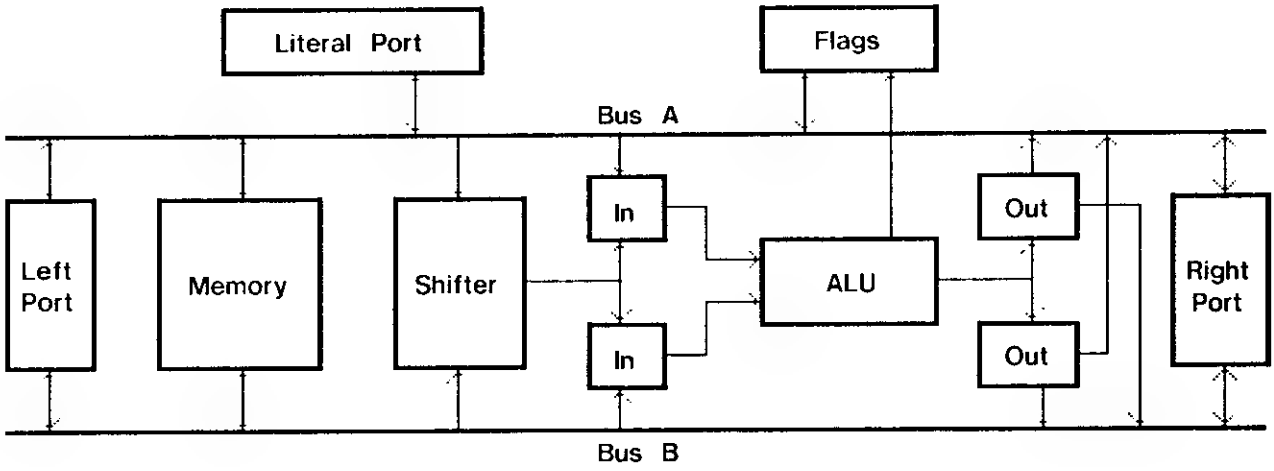


Figure 2. Block Diagram of OM2

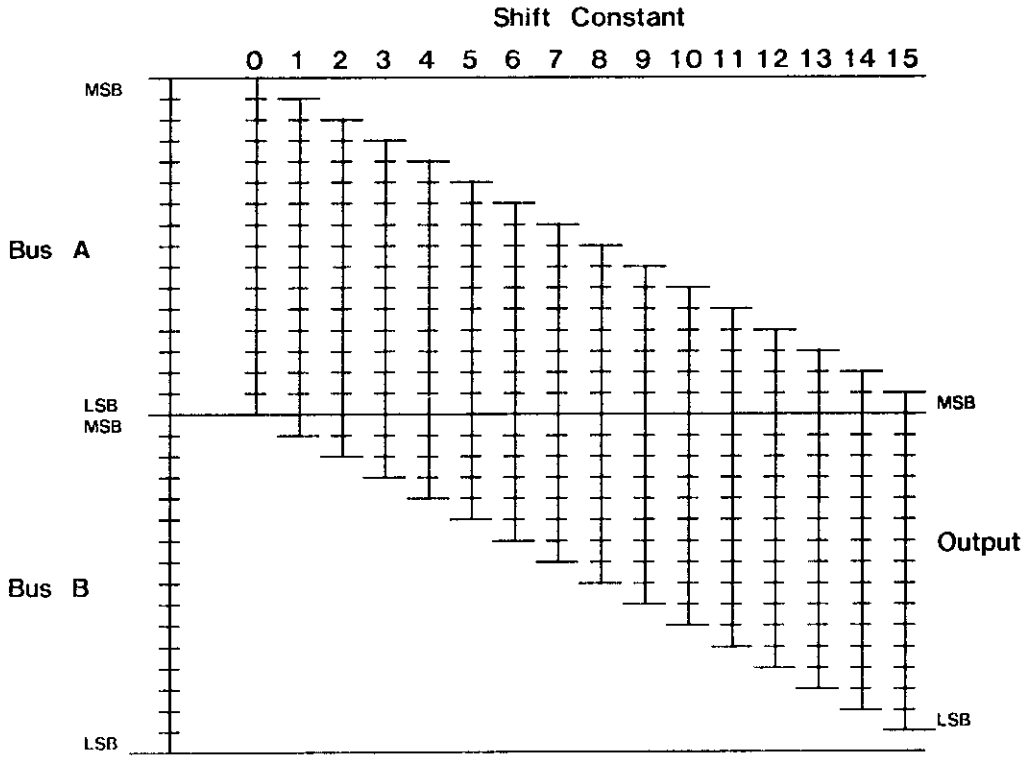


Figure 3. Shifter Operation.

The Left and Right Ports communicate between the datachip and the outside world. The Right Port has been traditionally known as the memory bus port while the Left Port has been the system bus port, but since the two ports are identical, this is an arbitrary convention. Each port has both an input latch and an output latch to provide facilities for synchronizing the datachip to the outside buses. Under program control either of the two buses can load the output latch during $\varphi 1$. There are three modes of driving data from the output latch to the pins two of which are under program control and one of which is under hardware control. The first method is to output the data as soon as it comes from the bus, during the same $\varphi 1$. The second method is to latch the data from the bus during $\varphi 1$ and drive it out during the following $\varphi 2$. The final method is to latch the data from the bus during $\varphi 1$, but output the data when an enable pin is pulled low. The enable pin would be controlled by a bus manager, and can be asynchronous with respect to the datachip. Inputting from the port is similar. By pulling down on another enable pin, data from the external bus is loaded into the input latch, which can be read later under program control. Alternatively, the microcode can force the data currently on the external bus into the internal bus during the current $\varphi 1$. With this scheme, many types of synchronous and asyncronous buses may be interfaced to OM2s. For internal control only, the external enable pins can be left floating.

Registers

The registers are static and dual port. Any one of the 16 registers may source either or both of the buses, while any one of the 16 may be the destination for either bus, but not both. There are only two restrictions to the use of the registers:

1. One register may not be the destination for both buses on the same cycle, and
2. One register may not be both the source for one bus and the destination for the other bus on the same cycle.

Shifter

The shifter concatenates the two buses, resulting in a 32-bit word, with the A bus being the more significant half. The shift constant then selects the bit position where the 16-bit output window starts. The shift constant specifies the number of bits from the B bus present in the output (ie. a shift constant of 0 returns the A bus, while a shift constant of 15 returns the LSB of the A bus in the MSB of the output, followed by all but the LSB of the B bus in the rest of the word). A conceptual picture of the shifter is shown in figure 3.

The ALU can select as inputs either the bus, the shift output, or shift control. If shift control is selected, the entire word is 0 except where the LSB of the A bus appears in the shift output. The shifter operates on $\varphi 1$; it may be viewed as an extension of the buses.

ALU

A block diagram of a single bit of the ALU is shown in figure 4. The ALU operates on the data which is contained in its two input latches. Input latch A may be loaded from the A bus, the shifter output, or the shift control, while the input latch B may be loaded from the B bus, the shifter output, or the shift control.

The outputs of the two latches become the inputs to two function blocks which determine what will happen on the carry chain. Function block P determines whether the carry chain propagates, while K decides if it is to kill the carry. If neither are true, the carry chain generates a carry. Each function block has four control inputs, which, for the Propagate function block, are referred to as PFF, PFT, PTF, and PTT. If PFF is enabled, the P block output is high if both input latches are false (contain 0). Enabling PFT activates the output if input A is false and input B is true, and so on. If, for example, both PFF and PFT are enabled, the output is active if input A is false, regardless of the state of input B. To further illustrate the operation of the function blocks, consider addition. If both inputs contain a 1, the carry is to be generated, while if both inputs are 0, the carry is killed. If the two inputs are different, the carry is to be propagated (carry out←carry in). To do this operation, the kill output should be active if both inputs are false, so KFF is enabled. Both PFT and PTF should be enabled to propagate properly. Therefore, K=(KFF, KFT, KTF, KTT)=(1,0,0,0), and P=(PFF, PFT, PTF, PTT)=(0,1,1,0).

The result of the ALU is produced by the R function block, which has as inputs P block out and Carry in. For the addition example above, the output should be the exclusive-or of P and Cin, so R=(0,1,1,0). P, K, and R values for common ALU operations are listed in the programming section.

Two ALU output latches (A and B) can be loaded from the R block output; either one may later be used to source either bus.

Flags

The carry input to the LSB of the ALU is a logical combination of a flag bit and two control inputs. The two control inputs can force the carry in to be either 1 or 0, or they can select either flag or flag bar as the input.

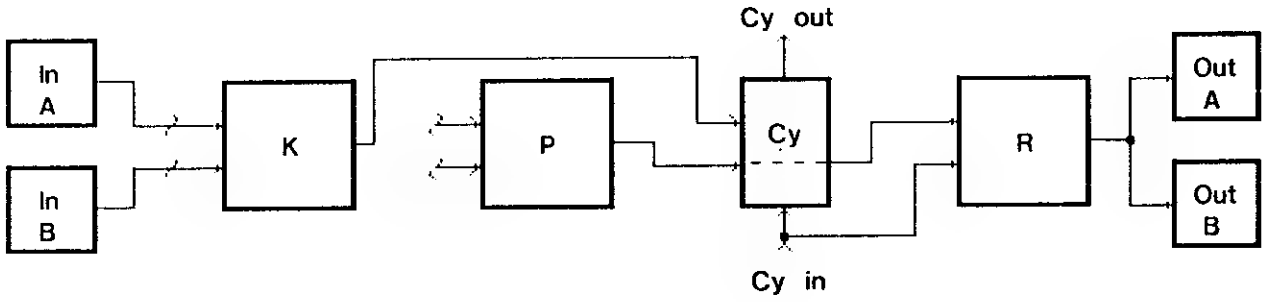


Figure 4. Block Diagram of one bit of the ALU

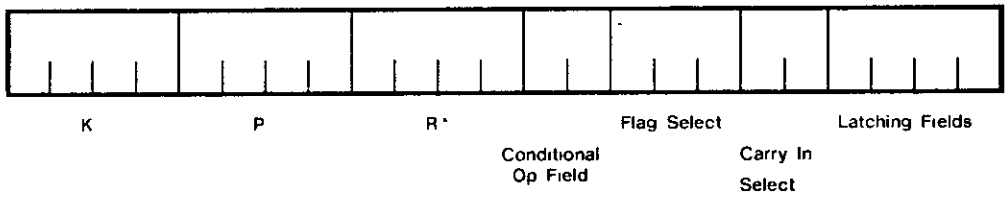


Figure 5a. Phi 2 Op Code (in on Phi 1)

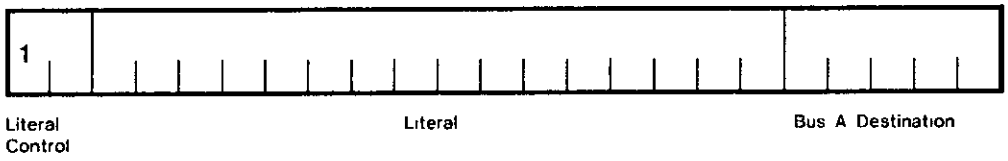


Figure 5b. Phi 1 Literal Transfer Op Code (in on Phi 2)

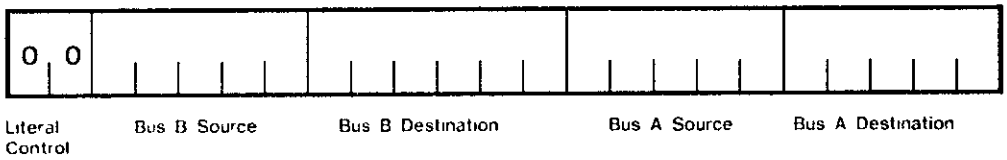


Figure 5c. Phi 1 Normal Op Code (in on Phi 2)

There is also a method for doing conditional ALU operations under the control of a two-bit conditional OP field. A conditional operation performed by the ALU is not only a function of the control inputs, but also of the flag bit. The conditional operation control forces some of the control inputs low, regardless of what the P, K, and R microcode says. The coding for conditional operations allows the use of operations like multiply step and divide step without the necessity for branching in the microcode.

There is a 16-bit flag register which can also be a source or destination of the A bus. This register can also be loaded with the ALU flags during $\varphi 2$. The ALU flags include *carry out, overflow, carry in to the MSB, zero, MSB, LSB, Less than, Less than or equal to,* and *Higher* (in unsigned value). The last three flags are comparison flags used after a subtaction. For example, after subtracting ALU input latch B from ALU input latch A, the "less than" flag is true if the value contained in ALU input latch B was larger than the value in ALU input latch A.

The MSB of the flag register is called the flag bit, and this bit may be modified every $\varphi 1$ by loading it with the value of one of the other bits of the flag register. The flag bit is used in the calculation of carry in and modification of conditional ALU Ops. This bit is also sent to the controller chip to be used for conditional branching, etc.

Literal

The one remaining datapath is the literal port. It is used to send data from the datachip to the controller, and vice versa. It is a source or destination for the A bus. When the literal port is being used, standard bus operations are suspended for that cycle.

Programming

The Datachip requires 23 bits of microcode on each phase of the clock. This section of the memo specifies the encoding of the fields within that microcode. Figure 5 shows the arrangement of the microcode word.

Bus Transfer

The bus transfer control bits enter the datachip during $\varphi 2$ and are used during the following $\varphi 1$. There are two buses, the A bus and the B bus, which interconnect the modules of the Datachip. These two buses are similar in many respects; however, there are a few asymmetries as to sources and destinations. Also, when a literal is being transferred, the only bus transfer field which is active is the A bus destination, which stores the literal entered on the A bus.

A listing of the bus sources and destinations follows:

	A Bus Source
0nnnn	Register n
10000	Right port pins
10001	Right port latch
10010	Left port pins
10011	Left port latch
10100	ALU output latch A
10101	ALU output latch B
10110	Flag register

	A Bus Destination
0nnnn	Register n
10000	Left port, drive now
10001	Left port, drive φ2
1001x	Left port, no drive
10100	Right port, drive now
10101	Right port, drive φ2
1011x	Right port, no drive
11000	ALU input latch A
11001	ALU input latch A gets shift out
11010	ALU input latch A gets shift control
11011	Flag Register

	B Bus Source
0nnnn	Register n
10000	Right port pins
10001	Right port latch
10010	Left port pins
10011	Left port latch
10100	ALU output latch A
10101	ALU output latch B

	B Bus Destination
00nnnn	Register n
010000	Left port, drive now
010001	Left port, drive φ2
01001x	Left port, no drive
010100	Right port, drive now
010101	Right port, drive φ2
01011x	Right port, no drive
0110xx	ALU input latch B
10nnnn	ALU input latch B gets shift output, shift const.=n
11nnnn	ALU input latch B gets shift control, shift const.=n

ALU Input Selection

The two ALU input latches are destinations for the two buses, as shown in the Bus Transfer section above. In addition to being loaded directly from the buses, these two latches can be loaded from the outputs of the shift array. The shift constant always comes from the 4 least significant bits of the B Bus Destination field, even though the destination of the B Bus is not the ALU input latch B. For example, the B Bus may be transferring the

contents of register 3 into register 5 while the A Bus is transferring the contents of register 4 to the ALU input atch A through the shifter. In this case, the shift constant would be "5", because the 4 least significant bits of the B Bus Destination field contain "0101".

ALU Operations

The following table shows coding for ALU operations that are commonly found useful. The user is encouraged to encode other operations if these are not suitable.The numbers given are the decimal representation of the 4 bit control word. For P and K, A'B'=1,A'B=2,AB'=4,AB=8. For R, P'C'=1,P'C=2,PC'=4,PC=8. **Cin** is the cary in select, and **Cond** .s the conditional OP select.

	K	P	R	Cin	Cond	
A+B	1	6	6	0	0	Add
A+B+Cin	1	6	6	1	0	Add with carry
A-B	2	9	6	2	0	Subtract
B-A	4	9	6	2	0	Subtract reverse
A-B-Cin	2	9	6	1	0	Subtract with borrow
B-A-Cin	4	9	6	1	0	Subtract rev. w/borrow
-A	12	3	6	2	0	Negative A
-B	10	5	6	2	0	Negative B
A+1	3	12	6	2	0	Increment A
B+1	5	10	6	2	0	Increment B
A-1	12	3	9	2	0	Decrement A
B-1	10	5	9	2	0	Decrement B
A∧B	0	8	12	0	0	Logical And
A∨B	0	14	12	0	0	Logical Or
A⊕B	0	6	12	0	0	Logical Exor
¬A	0	3	12	0	0	Not A
¬B	0	5	12	0	0	Not B
A	0	12	12	0	0	A
B	0	10	12	0	0	B
Mul	1	14	14	0	1	Multiply step
Div	3	15	15	0	2	Divide step
A/O	0	14	12	0	3	Conditional And/Or
Mask	10	5	8	2	0	Generate mask

Carry In Select

The Carry in select field determines what the carry into the LSB of the ALU will be, according to the following table:

00	0
01	Flag bit
10	1
11	Flag bit complemented

Conditional Op Select

The conditional op select field is used to generate 3 basic conditional type operations: Multiply, Divide, and And/Or step. In a great many cases, the conditional op allows functions dependant on a flag to be performed in one cycle, rather than sending the flag to the controller and branching to two separate instructions depending upon that flag. When a conditional OP is selected, certain ALU control bits are forced to zero. Which bits are zeroed depends on the conditional OP selectand the flag bit, as follows:

Select	Flag bit	K	P	R	
0	x	----	----	----	Unconditional
1	0	---0	--0-	--0-	Multiply step
	1	----	0---	0---	
2	0	0--0	-00-	-00-	Divide step
	1	-00-	0--0	0--0	
3	0	----	----	----	And/Or
	1	----	-00-	----	

Flags

The flag select field determines which of the ALU flags becomes the new flag bit. The following table lists the selection options.

Select	New Flag Bit
0	Old flag bit
1	Carry out
2	MSB
3	Zero
4	Less than
5	Less than or equal
6	Higher (in absolute value)
7	Overflow

The ALU flags are loaded into the flag register under the control of the latching field, bit 3. They are loaded into the following positions:

Bit		Flag
0		Not changed
1		Not changed
2		Not changed
3		Not changed
4		Not changed
5		Previous value of Flag bit
6	¬	Carry into MSB stage
7		Less than or equal
8	¬	Higher (in absolute value)
9	¬	Less than
10		LSB
11	¬	Zero
12		MSB
13		Overflow
14	¬	Carry out
15		Current Flag bit

Latching Field

The latching field specifies which of four registers should be loaded, as shown in the following table.

Latching Field	Register Loaded
1xxx	Flag register loaded with current AL flags
x1xx	ALU output atch A loaded with the ALU output
xx1x	ALU output latch B loaded with the ALU output
xxx1	The Literal field during the next φ2 is loaded with the contents of the A Bus during the last φ2
0000	None of these registers are affected

Literals

The two bit literal field specifies when a literal is to be used and which direction it goes. If both bits are 0, no literal transaction will occur. If the first bit is 1, a literal will be transfered. If the second bit is 1, the literal goes off chip, while if the bit is 0, the literal comes on chip.

Programming Examples

This section of the memo contains 3 programming examples which should provide a better understanding of the various datapaths within OM2.

The first example is 16-bit integer multiplication. The two inputs, X and Y, are multiplied to produce the result, Z. In the multiply loop, the number X is shifted left and the MSB is stripped off. Z is shifted left, then Y is added to the new Z if the MSB of X was a 1. The sequence of instructions is repeated 16 times, using the counter in the controller to signal when the 16 iterations have been performed. Figure 6 illustrates each step of the loop, which is listed here:

```
φ2:   ALU.Out.A←ALU(Shift left)←ALU.In.A;
      Latch Flags;
φ1:   ALU.In.A←Shift.out, Bus.A←ALU.Out.B;
      Bus.B←R[1];        ←This gives a shift constant of 1.
φ2:   ALU.Out.B←ALU(Multiply Step);      ←conditionally add.
      Flag←Cout;
φ1:   ALU.In A←Bus.A←ALU.Out.A
```

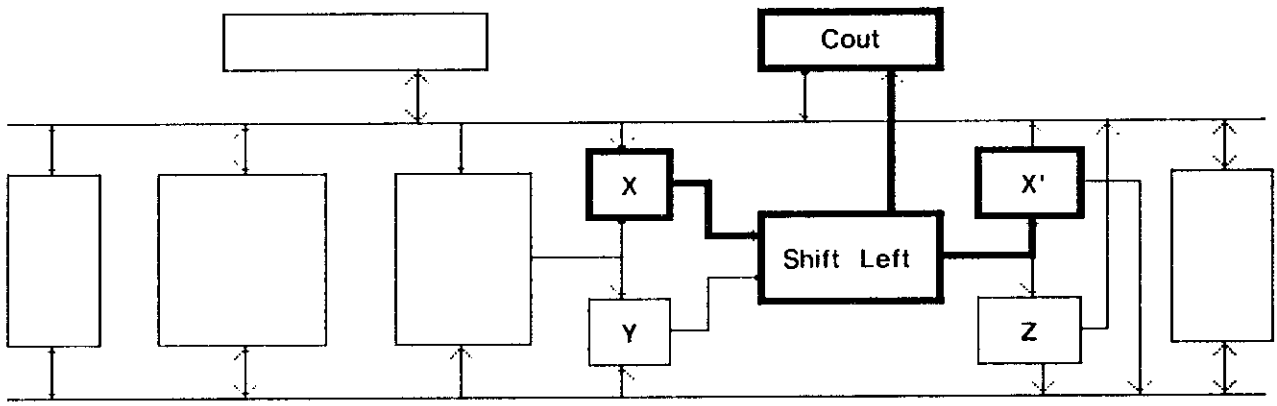


Figure 6a. Shift X in the ALU, putting the Cout flag into Flagbit. (Phi 2)

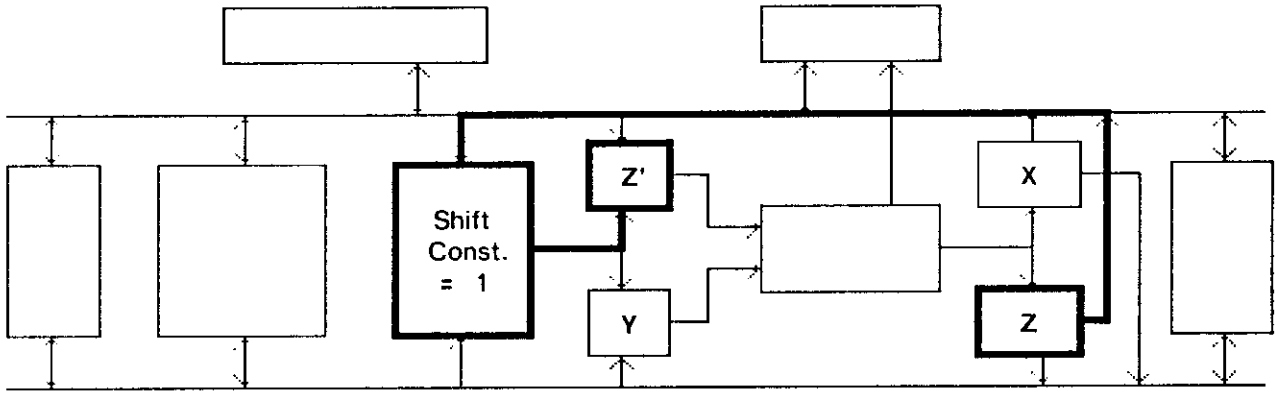


Figure 6b. Put Z on Bus A, and shift 1 left in shifter. (Phi 1)

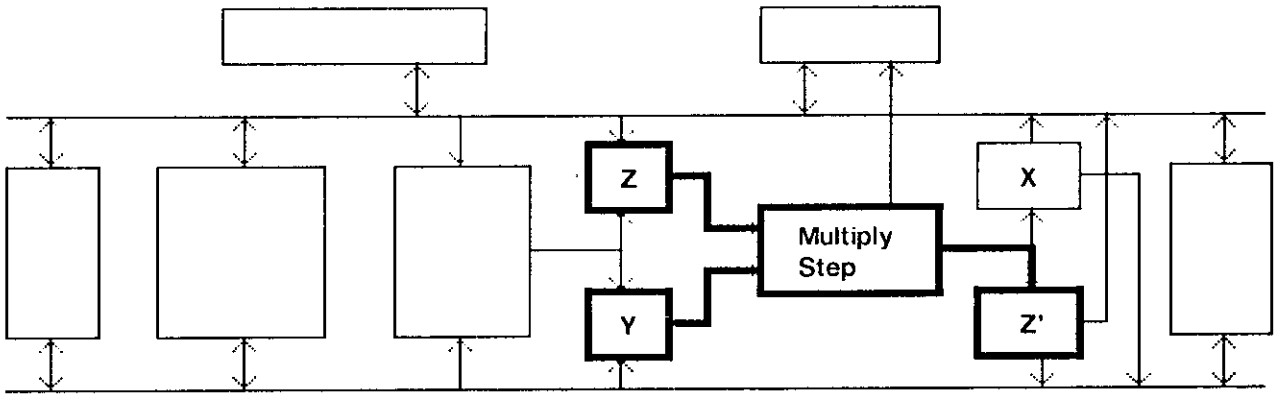


Figure 6c. Conditionally add Z and Y. (Phi 2)

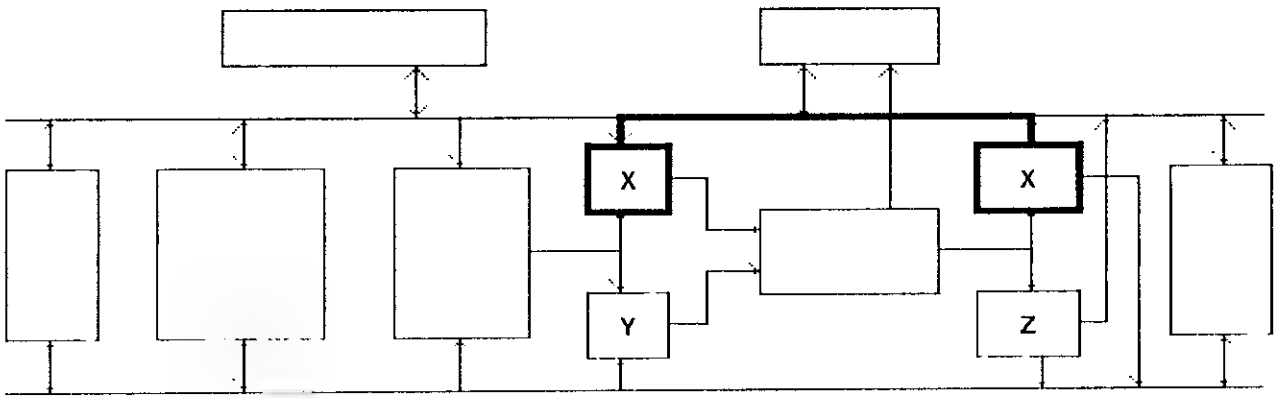


Figure 6d. Bring X back around to the ALU input. (Phi 1)

The second example will be to generate a parity flag, which is not directly available from the ALU. Parity is generated by exclusive-oring all of the bits of the data together. If the data are loaded into both ALU inputs, with the B input rotated by 1, performing an exclusive-or operation will give an output that is the exclusive-or of adjacent bits; bit *i* of the output will be bit *i* of the input ⊕ bit *i-1* of the same input. If this same operation is performed, this time rotating the B input by 2, bit *i* becomes *i* ⊕ *i-1* ⊕ *i-2* ⊕ *i-3*. By doing this 2 more times, rotating B first by 4 and then by 8, every bit of the output is equal to the parity: the exor of all of the bits. The MSB flag is the Parity Odd flag, while the Zero flag is the Parity Even flag. The program is listed here, and illustrated in figure 7:

```
φ1:  ALU.In.A←Bus.A←R[0];          ←generate the parity of register 0.
     ALU.In.B←Shift.out(1); Bus.B←R[0];
φ2:  ALU.Out.A←ALU(Exor);
φ1:  ALU.In.A←Bus.A←ALU.Out.A;
     ALU.In.B←Shift.out(2); Bus.B←ALU.Out.A;
φ2:  ALU.Out.A←ALU(Exor);
φ1:  ALU.In.A←Bus.A←ALU.Out.A;
     ALU.In.B←Shift.out(4); Bus.B←ALU.Out.A;
φ2:  ALU.Out.A←ALU(Exor);
φ1:  ALU.In.A←Bus.A←ALU.Out.A;
     ALU.In.B←Shift.out(8); Bus.B←ALU.Out.A;
φ2:  ALU(Exor);
```

The third example adds all of the registers to what is in ALU.Out.A. By executing and modifying a literal, the registers can be indirectly accessed, which makes this routine possible. Figure 8 illustrates the operation of the following code:

```
φ1:  ALU.In.A←Literal "Bus.A←R[1]; ALU.In.B←Bus.B←ALU.Out.B";
φ2:  ALU.Out.B←ALU←ALU.In.A;
φ1:  ALU.In.A←Bus.A←R[0];
φ2:  ALU.Out.B←ALU←ALU.In.A;       ←This is just setup, now the loop!
φ1:  Bus.A←ALU.Out.B;
     ALU.In.B←Bus.B←ALU.Out.A;
φ2:  ALU.Out A←ALU(add);
     Execute Literal;
φ1:  ALU.In.A←A.Bus;               ←The rest of this instruction is the literal!
φ2:  ALU.Out.B←ALU(increment)←ALU.In.B;     ←point to next register
```

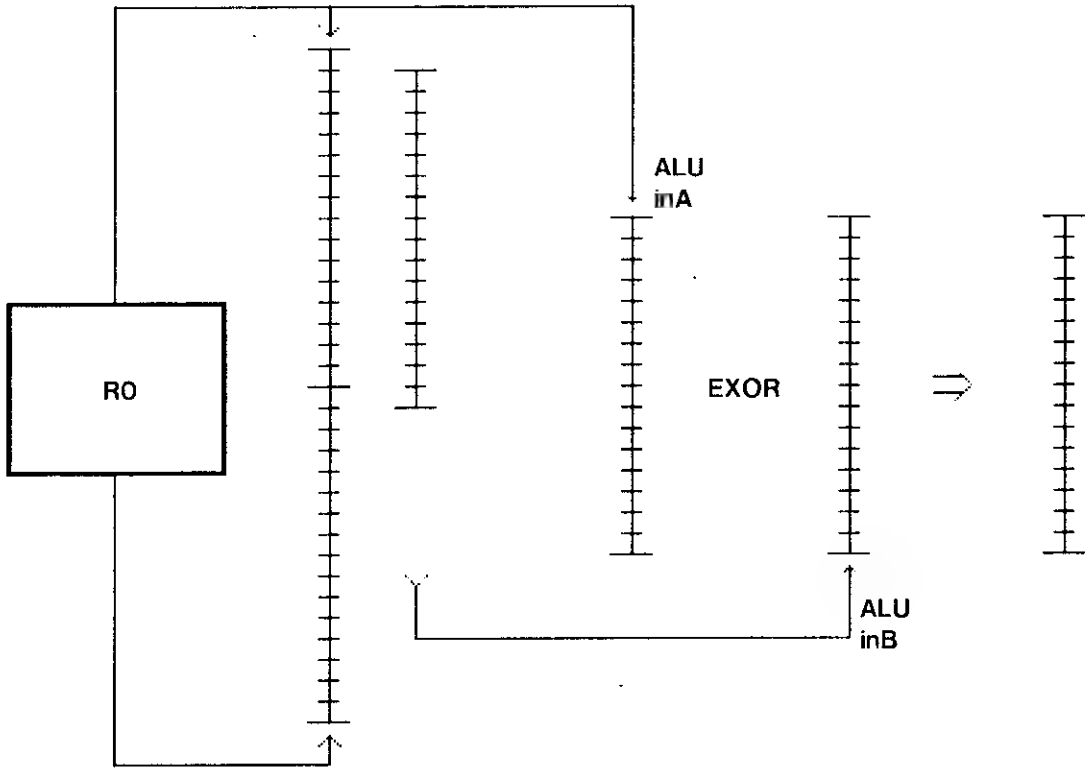


Figure 7a. Shifting by 1: Result is Exclusive-Or of Adjacent Bits.

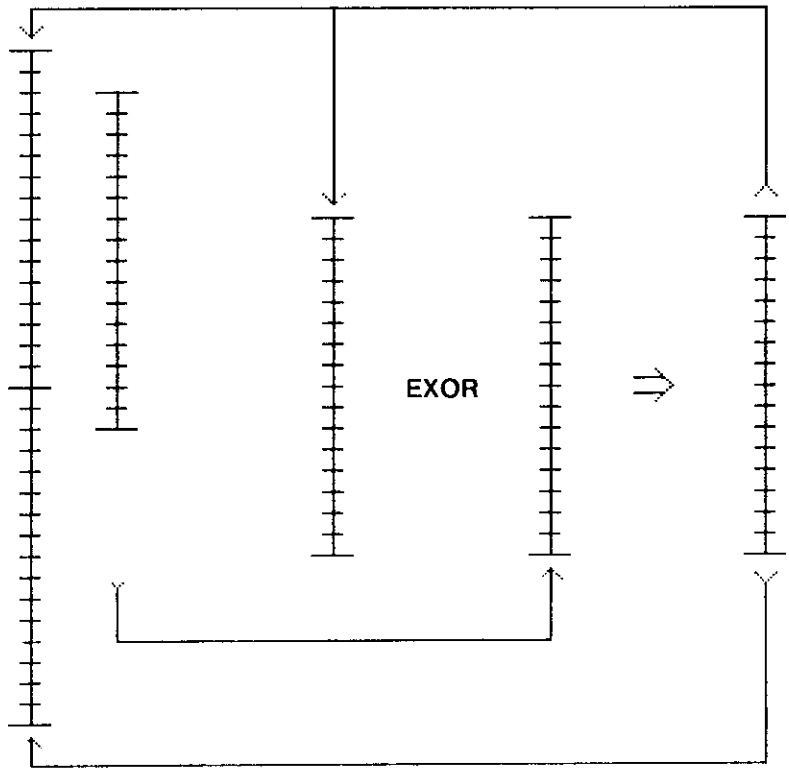


Figure 7b. Shifting by 2: Result is Exclusive-Or of 4 Adjacent Bits

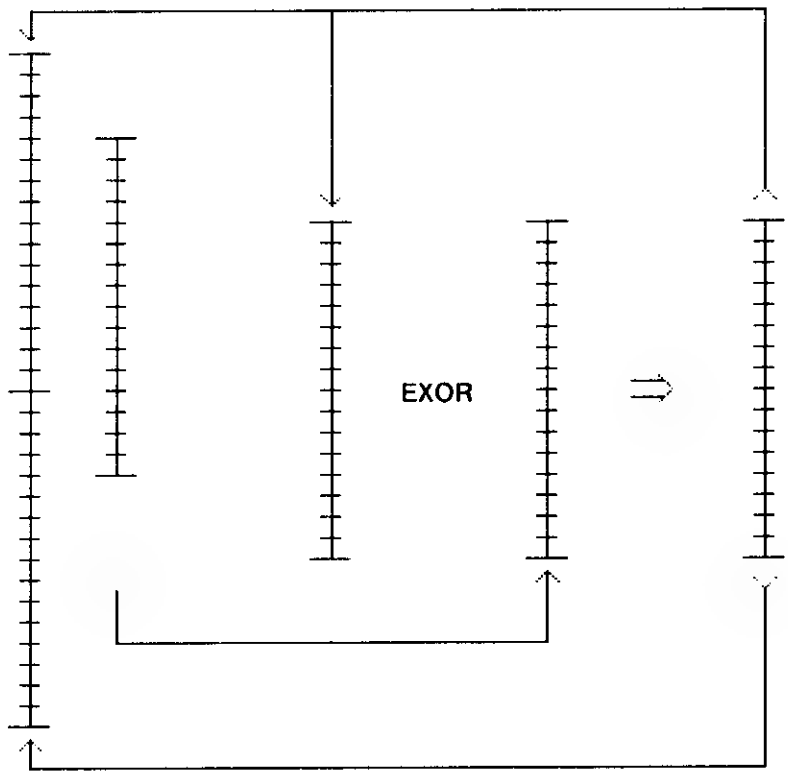


Figure 7c. Shifting by 4: Result is Exclusive-Or of 8 Adjacent Bits.

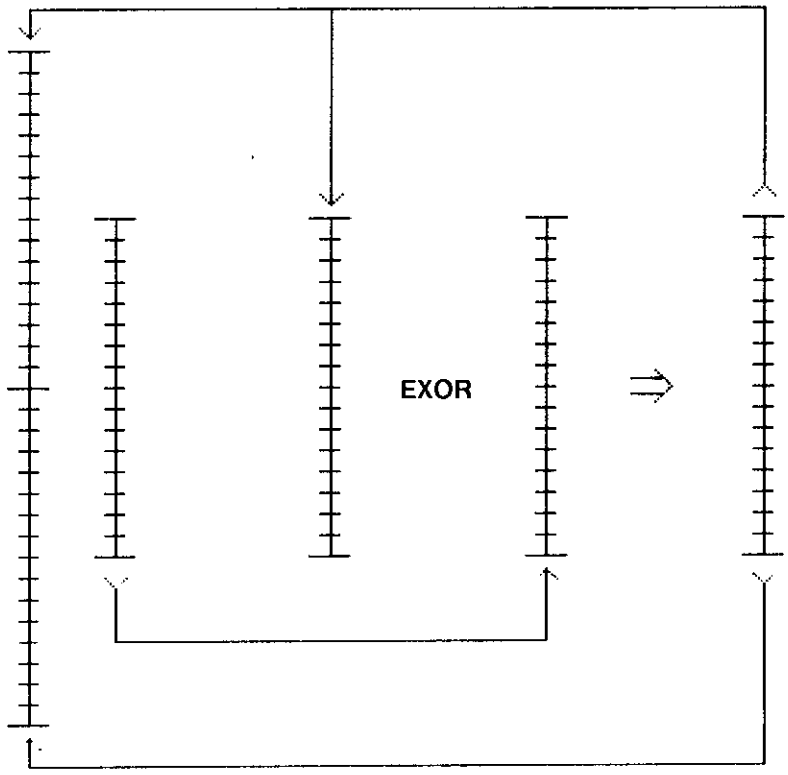


Figure 7d. Shifting by 8. Result Has All Bits Identically the Parity Flag.

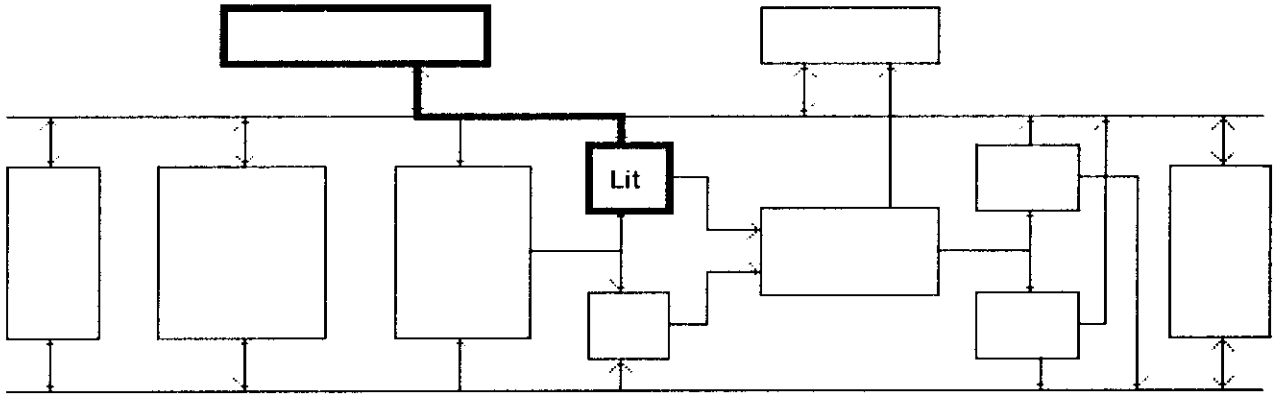


Figure 8a. Bring in Control Literal

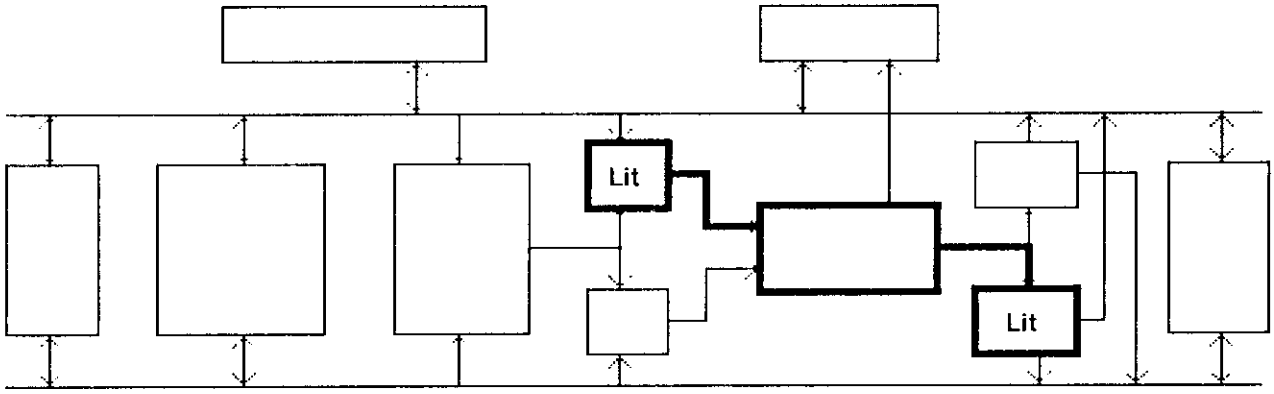


Figure 8b. Store in ALU.Out.B

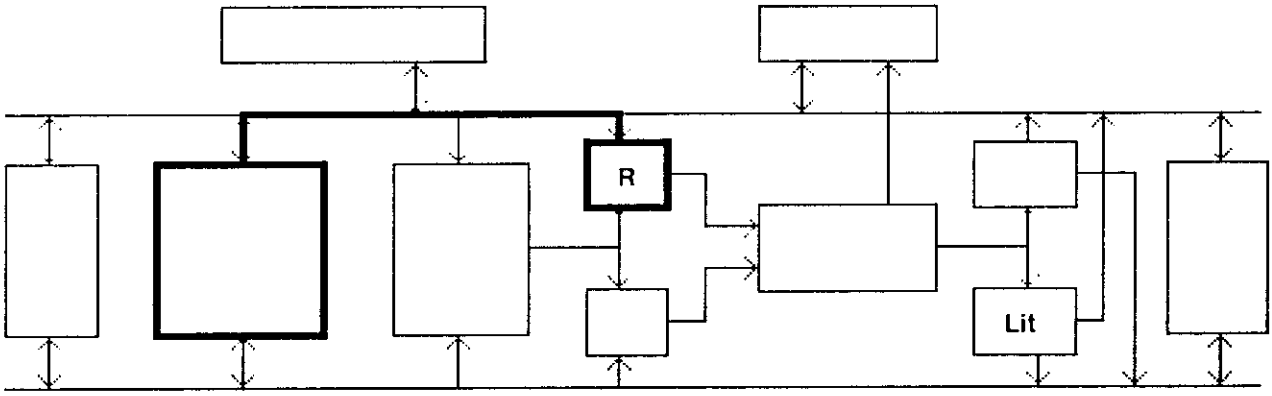


Figure 8c. Fetch Register 0

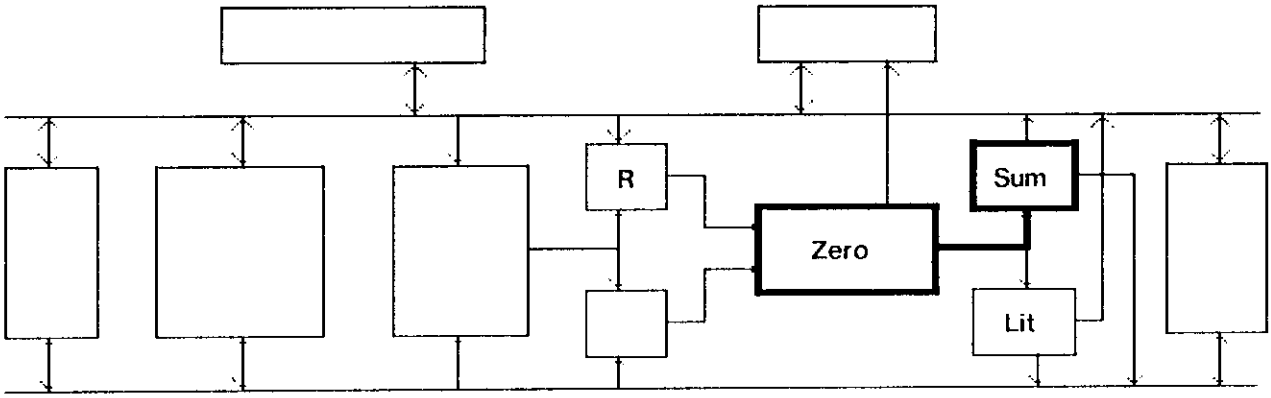


Figure 8d. Clear Sum

(fig8abcd sil)

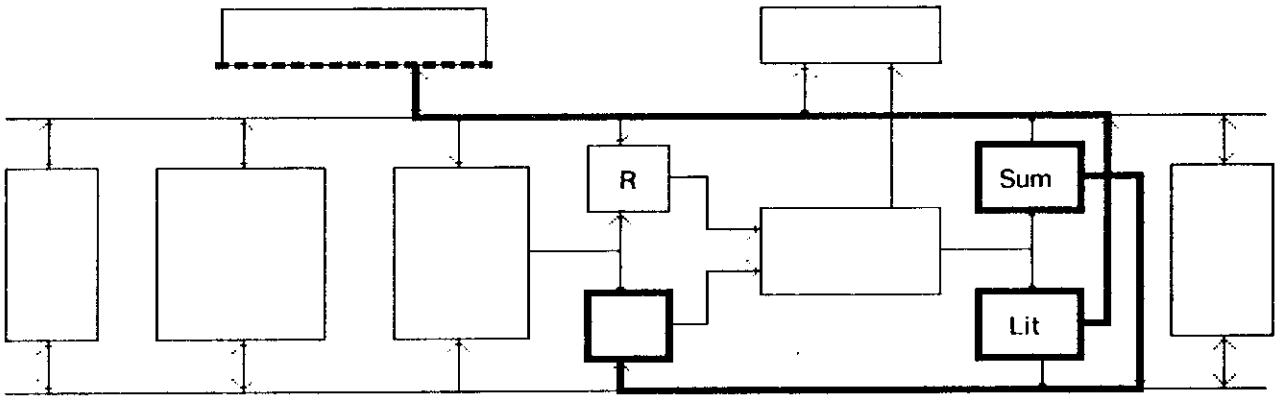


Figure 8e. Bring Around Sum and Put Control Literal on Bus A

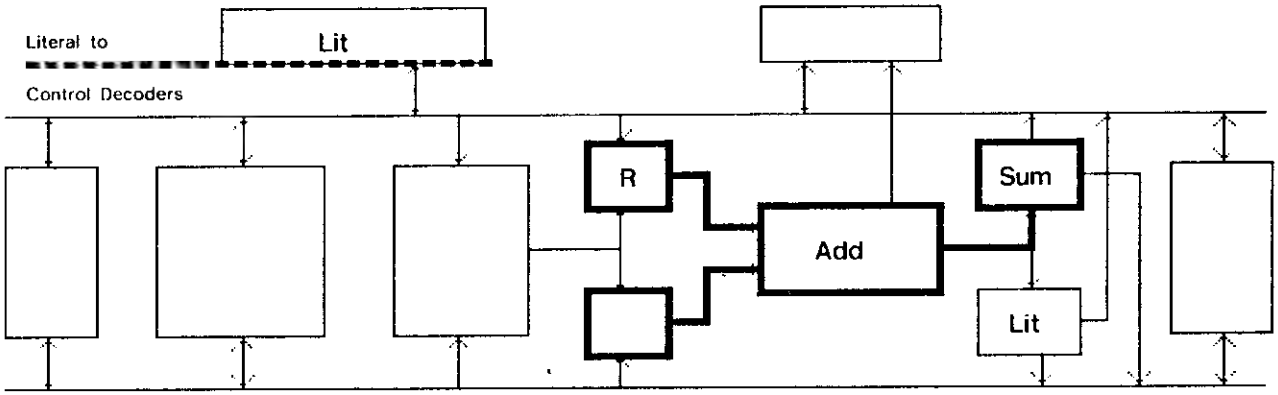


Figure 8f. Add Current Numbers

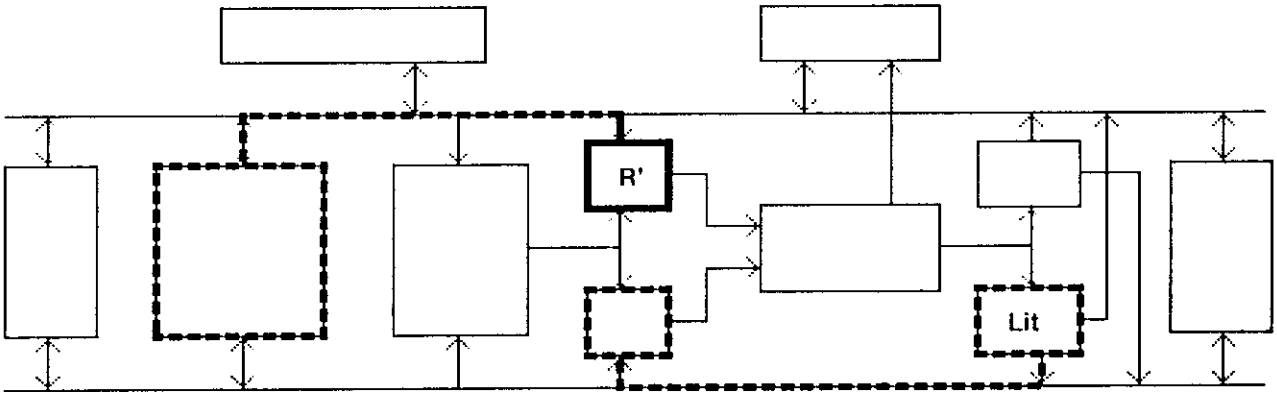


Figure 8g. Register Loaded by Literal Goes to ALU Input A

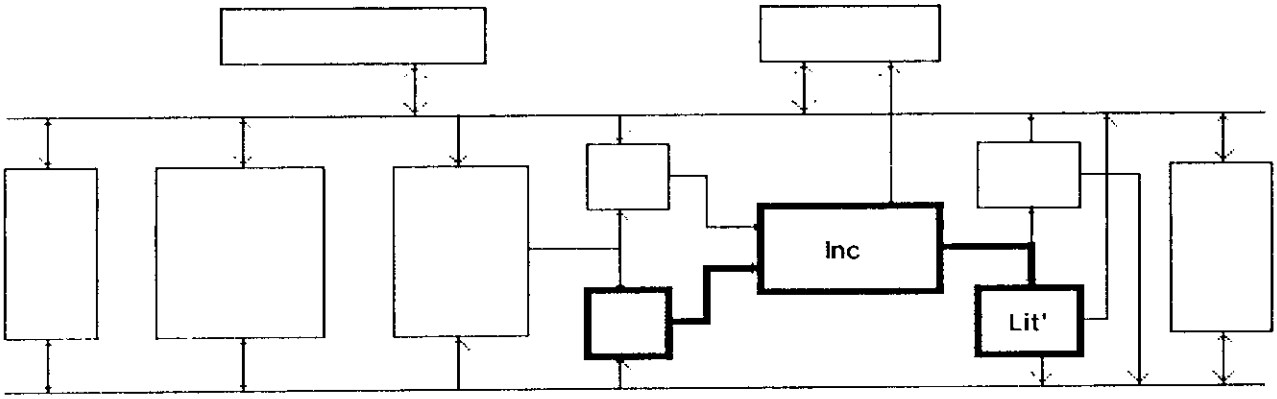


Figure 8h. Point to Next Register, Loop to Figure 8e (fig8efgh sif)

ISP Description of the OM2 Datachip[1]

P.n States

lp<0:17>	*left port*
rp<0.17>	*right port*
new.code<0:22>	*microcode*
flag.pin<0>	*flag to controller*
power<0:3>	*power, ground, clock, substrate*

Pin Formats

left port.data<0:15>	:= lp<0:15>
left.out.async<0>	:= lp<16>
left.in async<0>	:= lp<17>
right port.data<0:15>	:= rp<0:15>
right.out.async<0>	:= rp<16>
right.in async<0>	:= rp<17>
literal<0:15>	:= new code<5:20>
clock<0>	:= power<3>

Mp State

reg[0:15]<0:15>	*registers*
a.bus<0:15>	*bus a*
a.bus.old<0:15>	*bus a latched for a literal*
b.bus<0:15>	*bus b*
left.out<0:15>	*left pad output latch*
left.in<0:15>	*left pad input latch*
right out<0:15>	*right pad output latch*
right.in<0:15>	*right pad input latch*
left.out.later<0>	*for output during φ2 operations*
right.out.later<0>	*for output during φ2 for right port*
alu.in.a<0:15>	*alu input latch a*
alu.in.b<0:15>	*alu input latch b*
alu.out.a<0:15>	*alu output latch a*
alu.out.b<0:15>	*alu output latch b*
old.code<0:22>	*microcode that came in last phase*
flags<0:15>	*flag register*

Instruction format

a.source<0:4>	:= old.code<5:9>
b.source<0:4>	:= old.code<16:20>
a.destination<0:4>	:= old.code<0:4>
b.destination<0:5>	:= old.code<10:15>
literal.in<0>	:= old.code<22>
old.literal<0:15>	:= old.code<5:20>
alu.p.op<0:3>	:= old.code<19:22>
alu.k.op<0:3>	:= old.code<15:18>
alu.r.op<0:3>	:= old.code<11:14>
alu.conditional<0:1>	:= old.code<9:10>
flag.select<0:2>	:= new.code<6:8>
carry.in.select<0:1>	:= old.code<4:5>
latch flags<0>	:= old.code<3>
latch alu.out.a<0>	:= old.code<2>
latch alu.out.b<0>	:= old.code<1>
literal.control<0>	:= old.code<0>
reg.select.1<0.3>	:= a.source<0:3>
reg select 2<0.3>	:= a destination<0:3>

```
    reg.select.3<0:3>                   := b.source<0:3>
    reg.select.4<0:3>                   := b.destinat on<0:3>
    select.1<0>                         := a.source<4>
    select.2<0>                         .= a destination<4>
    select 3<0>                         := b.source<4>
    select.4<0:1>                       := b.destination<4:5>
    shift.constant<0:3>                 := b.destination<0:3>
    sharay<0:31>                        .= b.bus<0:15>□a.bus<0:15>


Temporary State
    kill.control<0:3>
    propagate.control<0:3>
    result.control<0:3>
    kill<0:15>
    propagate<0:15>
    carry<0:16>
    alu.out<0:15>


Instruction Execution
Instruction.execution:=(
    left.out async=0⇒(left.port data←left.out);next
    left.in.async=0⇒(left in← eft.port.data),next
    right.out.async=0⇒(right.port.data←right.out);next
    right in.async=0⇒(right.in←right.port.data);next
    phi1(.=clock=1)⇒(
        left.out.later←0;next
        right.out.later←0;next
        l.teral.in=1⇒(a.bus←old.literal);next
        literal.in=0⇒(
            select.1=0⇒(a.bus←reg[reg.select.1]);
            select.1=1⇒(
                reg.select.1=0⇒(a.bus←right.in←right.port.data);
                reg.select.1=1⇒(a.bus←right.in);
                reg.select.1=2⇒(a.bus←left.in←left.port.data);
                reg.select.1=3⇒(a.bus←left.in);
                reg.select.1=4⇒(a.bus←alu.out.a);
                reg select.1=5⇒(a.bus←alu.out.b);
                reg select.1=6⇒(a bus←f ags);next);next
            select.3=0⇒(b.bus←reg[reg.select.3]);
            select.3=1⇒(
                reg select.3=0⇒(b.bus←right.in←right.port.data);
                reg.select.3=1⇒(b.bus←right.in);
                reg.select.3=2⇒(b.bus←left.in←left.port.data);
                reg.select.3=3⇒(b.bus←left.in);
                reg.select.3=4⇒(b.bus←alu.out.a);
                reg.select.3=5⇒(b.bus←alu.out.b);next);next
            select.4=0⇒(reg[reg.select.4]←b.bus);
            select.4=1⇒(
                reg select.4=0⇒(left.port.data←left.out←b.bus);
                reg.select.4=1⇒(
                    left.out←b.bus;next
                    left.out.later←1;next);
                reg.select.4=2⇒(left.out←b.bus);
                reg select.4=3⇒(left.out←b.bus);
                reg select.4=4⇒(right.port.data←right.out←b.bus);
                reg select.4=5⇒(
                    right.out←b.bus;next
                    right.out.later←1;next);
```

```
            reg.select.4=6⇒(right.out←b.bus);
            reg.select.4=7→(right.out←b.bus);
            reg.select.4∈{8,9,10,11}⇒(alu.in.b←b.bus);next);
        select.4=2⇒(alu.in.b<0:15>←
                                    sharay<16-shift.constant:31-shift constant>);
        select.4=3⇒(alu.in.b←2↑shift.constant);next);next
    select.2=0⇒(reg[reg.select.2]←a.bus);
    select.2=1⇒(
        reg.select.2=0⇒(left.port.data←left.out←a.bus);
        reg.select.2=1⇒(
            left.out←a.bus;next
            left.out.later←1;next);
        reg.select.2=2⇒(left.out←a.bus);
        reg.select.2=3⇒(left.out←a.bus);
        reg.select.2=4⇒(right.port.data←right.out←a.bus);
        reg.select.2=5⇒(
            right.out←a.bus;next
            right.out.later←1;next);
        reg.select.2=6⇒(right.out←a.bus);
        reg.select.2=7⇒(right.out←a.bus);
        reg.select.2=8⇒(alu.in.a←a.bus);
        reg.select.2=9⇒(alu.in.a<0:15>←
                                    sharay<16-shift.constant:31-shift.constant>);
        reg.select.2=10⇒(alu.in.a←2↑shift.constant);
        reg.select.2=11⇒(flags←a.bus);next);next
    flag.select=1⇒(flags<15>←flags<14>);
    flag.select=2⇒(flags<15>←flags<12>);
    flag.select=3⇒(flags<15>←flags<11>);
    flag.select=4⇒(flags<15>←flags<9>);
    flag.select=5⇒(flags<15>←flags<7>);
    flag.select=6⇒(flags<15>←flags<8>);
    flag.select=7⇒(flags<15>←flags<13>);next


phi2(:=clock=0)⇒(
    left.out.later=1⇒(left.port.data←left.out);next
    right.out.later=1⇒(right.port.data←right.out);next
    kill.control←alu.k.op;next
    propagate.control←alu.p.op;next
    result.control←alu.r.op;next
    alu.conditional=1⇒(
        flags<15>=1⇒(
            propagate.control<0>←0;next
            result.control<0>←0;next);
        flags<15>=0⇒(
            kill.control<3>←0;next
            propagate.control<2>←0;next
            result.control<2>←0;next);next);
    alu.conditional=2⇒(
        flags<15>=1⇒(
            kill.control<2>←0;next
            kill.control<1>←0;next
            propagate.control<3>←0;next
            propagate.control<0>←0;next
            result.control<3>←0;next
            result.control<0>←0;next);
        flags<15>=0⇒(
            kill.control<3>←0;next
            kill.control<0>←0;next
            propagate.control<2>←0;next
```

```
                propagate.control<1>←0;next
                result.control<2>←0;next
                result.control<1>←0;next);next);
        alu conditional=3⇒(
            flags<15>=1⇒(
                propagate.control<2>←0;next
                propagate.control<1>←0;next);next);next
        kill<0:15>←(
            kill.control<3>∧(¬alu.in.a<0:15>)∧(¬alu.in.b<0:15>)∨
            kill.control<2>∧(¬alu.in.a<0:15>)∧alu.in.b<0:15>∨
            kill.control<1>∧alu.in.a<0:15>∧(¬alu.in.b<0:15>)∨
            kill.control<0>∧alu.in.a<0:15>∧alu.in.b<0:15>);next
        propagate<0:15>←(
            propagate.control<3>∧(¬alu.in.a<0:15>)∧(¬alu.in.b<0:15>)∨
            propagate.control<2>∧(¬alu.in.a<0:15>)∧alu.in.b<0:15>∨
            propagate.control<1>∧alu.in.a<0:15>∧(¬alu.in.b<0:15>)∨
            propagate.control<0>∧alu.in.a<0:15>∧alu.in.b<0:15>);next
        carry<0>←carry.in.select<1>⊕(carry.in.select<0>∧flags<15>);next
        for k=1 step 1 until 16 do:
            (carry<k>←¬(kill<k-1>+propagate<k-1>*¬carry<k-1>)+kill<k-1>*
                propagate<k-1>*x);next                    in OM2, x is
undefined
            If kill(i) and propagate(i) are both high, the carry chain does funny
things.
            We represent that here by use of the "x" in the carry function.
        alu.out<0:15>←(
            result.control<3>∧(¬propagate<0:15>)∧(¬carry<0:15>)∨
            result.control<2>∧(¬propagate<0:15>)∧carry<0:15>∨
            result.control<1>∧propagate<0:15>∧(¬carry<0:15>)∨
            result.control<0>∧propagate<0:15>∧carry<0:15>);next
        latch.alu out.a=1⇒(alu.out.a←alu.out);next
        latch.alu out.b=1⇒(alu.out.b←alu.out);next
        literal.control=1⇒(literal←bus.a.old);next
        latch.flags=1⇒(
            flags<5>←flags<15>;next
            flags<6>←carry<15>;next
            flags<10>←alu.out<0>;next
            flags<11>←0;next
            alu.out=0⇒(flags<11>←1);next
            flags<12>←alu.out<15>;next
            flags<14>←carry<16>;next
            flags<13>←flags<14>⊕flags<6>;next
            flags<9>←flags<12>⊕flags<13>;next
            flags<7>←flags<11>∨flags<9>;next
            flags<8>←¬(flags<14>∨flags<11>);next);next);next
                    )                    end of instruction execution
```

Reference:

1. C. G. Bell, A. Newell, "Computer Structures: Readings and Examples", Chap. 2, McGraw-Hill, 1971.

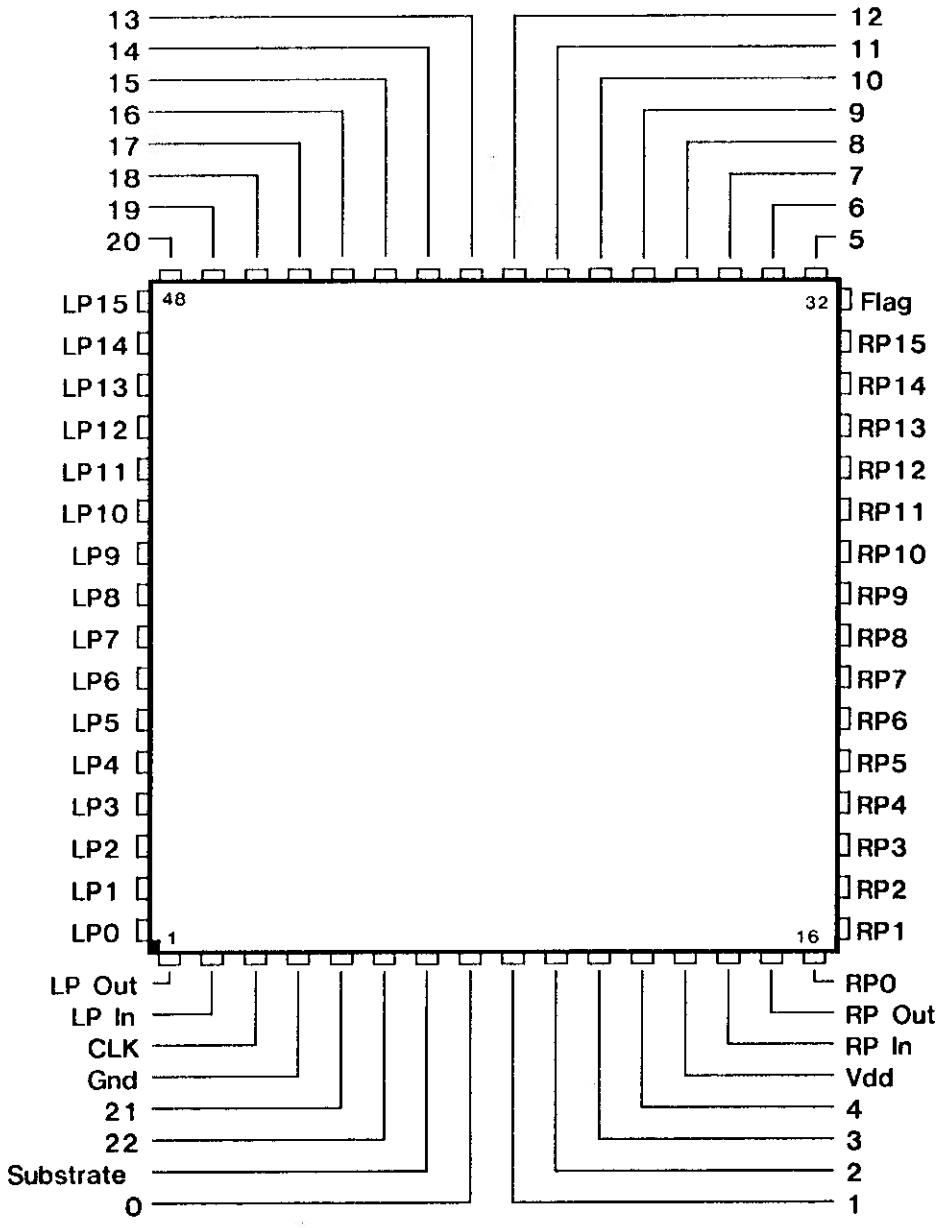
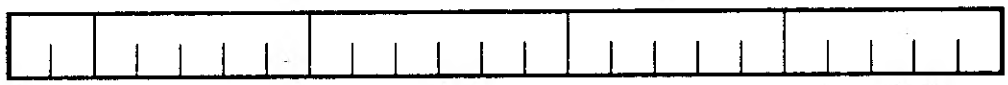


Figure 9. Pinout of the OM2 Datachip

Summary of Commands

Transfer Phase: PHI 1



Bus A Source

0nnnn	Register n
10000	Right Port Pins
10001	Right Port Latch
10010	Left Port Pins
10011	Left Port Latch
10100	ALU Output Latch A
10101	ALU Output Latch B
10110	Flag Register
-------	Literal (see Literal Control)
other	No Source

Literal Control

000	Microcode In
001	Illegal
010	Literal In
011	Illegal
100	Execute old A Bus
101	Illegal
110	A Bus gets old A Bus
111	Literal Out

└── LSB of the Latching Field during last PHI 2.

Bus B Source

0nnnn	Register n
10000	Right Port Pins
10001	Right Port Latch
10010	Left Port Pins
10011	Left Port Latch
10100	ALU Output Latch A
10101	ALU Output Latch B
other	No Source

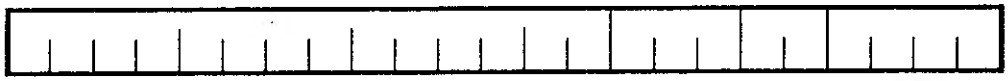
Bus A Destination

0nnnn	Register n
10000	Left Port, drive now
10001	Left Port, drive PHI 2
1001x	Left Port, no drive
10100	Right Port, drive now
10101	Right Port, drive PHI 2
1011x	Right Port, no drive
11000	ALU Input Latch A
11001	ALU Input Latch A gets Shift Out
11010	ALU Input Latch A gets Shift Control
11011	Flag Register
other	Do Destination

Bus B Destination

00nnnn	Register n
010000	Left Port, drive now
010001	Left Port, drive PHI 2
01001x	Left Port, no drive
010100	Right Port, drive now
010101	Right Port, drive PHI 2
01011x	Right Port, no dirve
0110xx	ALU Input Latch B
0111xx	No Destination
10nnnn	ALU Input Latch B gets shift output, shift constant=n
11nnnn	ALU Input Latch B gets shift control, shift constant=n

Operation Phase: PHI 2



ALU Operation

1000	0110	0110	00	00	Add
1000	0110	0110	00	01	Add with Carry
0100	1001	0110	00	10	Subtract
0010	1001	0110	00	10	Subtract Reversed
0100	1001	0110	00	01	Subtract with Borrow
0010	1001	0110	00	01	Subtract Reversed with Borrow
0011	1100	0110	00	10	Negative A
0101	1010	0110	00	10	Negative B
1100	0011	0110	00	10	Increment A
1010	0101	0110	00	10	Increment B
0011	1100	1001	00	10	Decrement A
0101	1010	1001	00	10	Decrement B
0000	0001	0011	00	00	Logical AND
0000	0111	0011	00	00	Logical OR
0000	0110	0011	00	00	Logical Exclusive Or
0000	1100	0011	00	00	Not A
0000	1010	0011	00	00	Not B
0000	0011	0011	00	00	A
0000	0101	0011	00	00	B
1000	0111	0111	01	00	Multiply Step
1100	1111	1111	10	00	Divide Step
0000	0111	0011	11	00	Conditional AND/OR
0101	1010	0001	00	10	Generate Mack
uuuu	uuuu	uuuu	uu	uu	User Defined Op

└── Carry In Select Field

Carry In Select

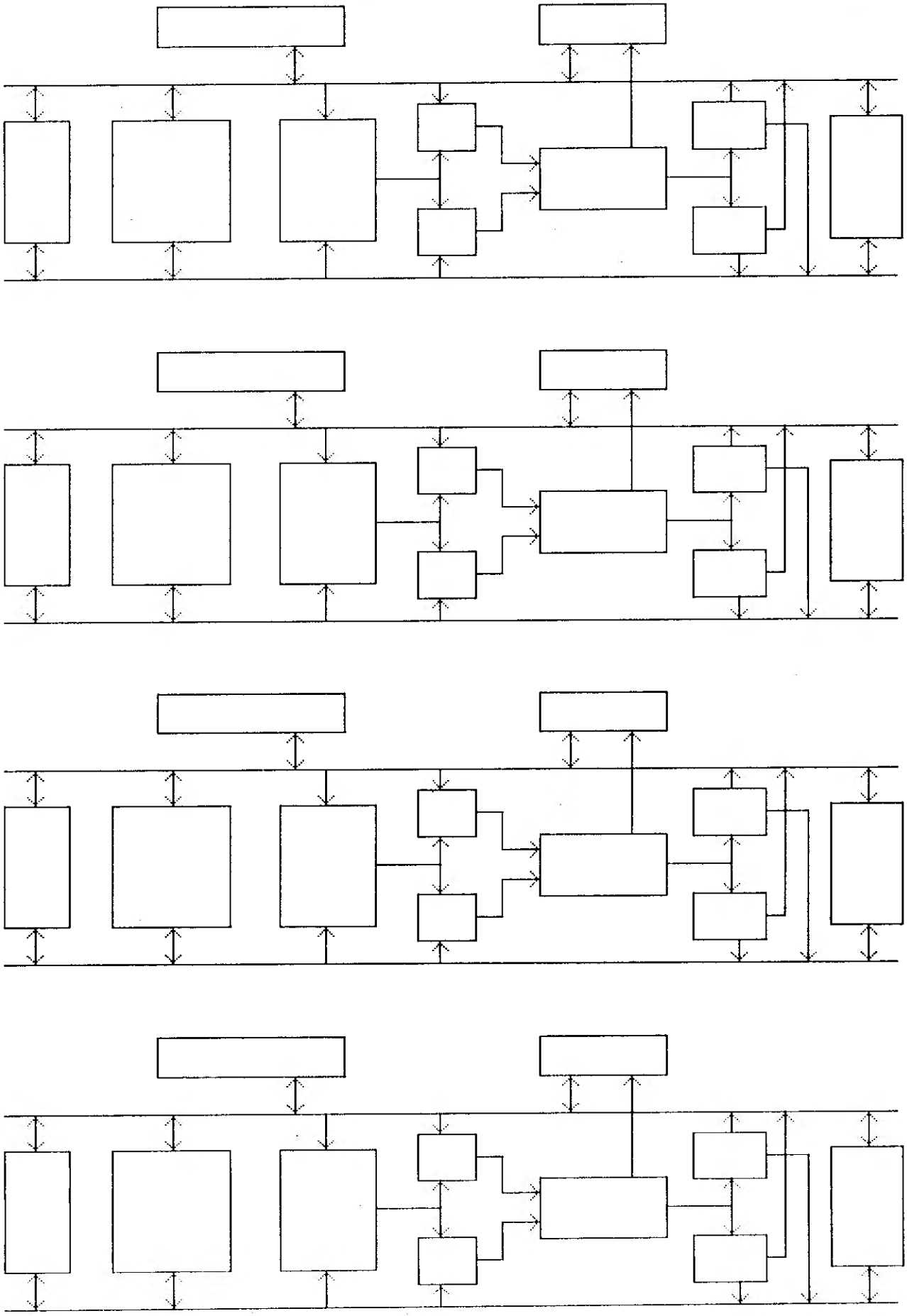
00	0
01	Flagbit
10	1
11	Flagbit Complimented

Flag Select

000	Old Flagbit
001	Carry Out
010	MSB
011	Zero
100	Less than flag
101	Less than or equal flag
110	Higher flag
111	Overflow

Latching Field

1xxx	Latch Flags
x1xx	Load ALU Output Latch A
xx1x	Load ALU Output Latch B
xxx1	Literal bits get old A Bus next PHI 1
0000	Nop



(fig11.sil)